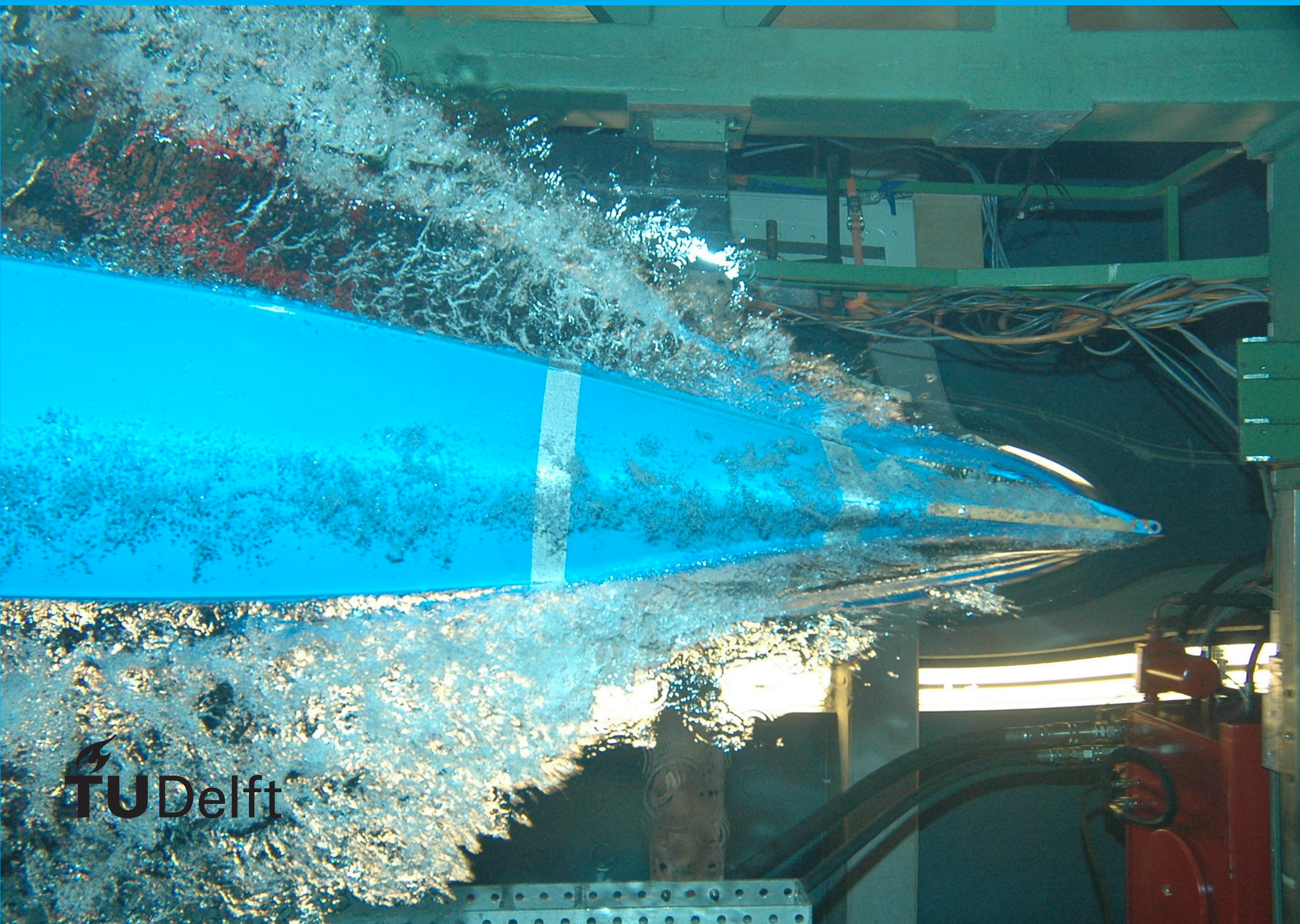
# Large-scale analysis in firmware images security using Embedded Binary Analysis Tool

## EBAT

Paris Panagiotou



# Large-scale analysis in firmware images security using Embedded Binary Analysis Tool

## EBAT

by

Paris Panagiotou

to obtain the degree of Master of Science in Embedded Systems
at the Delft University of Technology,
to be defended publicly on July 3, 2024.

| | | |
|---|---|---|
| Graduation committee: | Prof. Dr. Zekeriya Erkin | Delft University of Technology |
| | Prof. Dr. R. R. Venkatesha Prasad | Delft University of Technology |
| | Prof. Christian Doerr | External Supervisor |

TU Delft

# Preface

To my wife and family.

*Paris Panagiotou*
*Delft, July 2024*

# Abstract

This thesis researches the security of firmware images in the Internet of Things (IoT) and embedded devices. We present an open-source tool, Embedded Binary Analysis Tool (`EBAT`), designed to analyze cross-architectural firmware image security context. `EBAT` consists of various modules capable of discovering outdated software for various libraries, particularly on cryptographic libraries, and detecting Common Vulnerabilities and Exposures (CVEs), focusing on firmware's cryptographic libraries. It also detects exploit mitigation techniques on firmware's image binaries and discovers credentials and passwords with a focus on private keys embedded in the firmware image. Additionally, `EBAT` identifies Application Programming Interfaces (APIs) cryptographic misuses through static taint analysis (backward tracking) on cross-architectural binaries. We presented a total of $18$ well-defined cryptographic rules and a list of $733$ function calls with more than $1,600$ function arguments, applicable in static taint analysis to check the possibility of cryptographic misuses based on $10$ well-used open-source cryptographic libraries APIs. `EBAT's` static taint analysis provides a powerful framework for detecting the possibility of cryptographic misuses in cross-architectural binaries, making it a valuable tool for identifying and addressing vulnerabilities in cryptographic implementation in firmware images.

Using `EBAT`, we conducted a large-scale analysis of over $36,000$ firmware images publicly crawled from the Internet and successfully unpacked over $60\%$ of them. The created dataset of firmware images includes more than $5,000$ different products across $33$ vendors, spanning more than $20$ years and a plethora of various device types. Our findings show that *ARM and MIPS* are the most prevailed CPU architecture in the *IoT/embedded* industry. We compared identical binaries across all vendors, revealing a significant percentage of similar binaries used across different vendors' firmware images. Our analysis of firmware binaries reveals a notable absence of exploit mitigation techniques in *IoT/embedded* firmware images, and we present many firmware images containing private keys, posing potential security threats. Additionally, versions of open-source cryptographic libraries used in firmware images are identified, and the CVEs of the cryptographic libraries are evaluated. Two real-world case studies on hard-coded credentials demonstrate the significance of the large-scale attack presented in this thesis. Hashed passwords, predominantly using outdated algorithms, have also been discovered, and several have been cracked.

The main goal of `EBAT` is to identify cryptographic misuses in cross-architectural binaries. By applying static taint analysis (backward tracking) to well-defined APIs on specific functions and arguments for $10$ open-source cryptographic libraries, we can identify potential violations of cryptographic rules. This analysis was executed on over $1.4$ million binaries, revealing that approximately $50\%$ of examined firmware images violated at least one cryptographic rule. Various case studies on real-world vulnerabilities in firmware images are presented, including recent CVEs that are found in various vendors' products. Executing `EBAT` on those vulnerable firmware images, we tested the effectiveness of our tool to evaluate the automatic capturing of these known vulnerabilities. In addition, performing large-scale analysis on an extensive corpus of firmware images allows us to discover that other firmware images are affected by these known vulnerabilities, in some cases also across various product lines not covered on the public CVEs reports.

In conclusion, `EBAT` is a valuable resource for researchers working on firmware security. Its automated analysis process, comprehensive modules, and ability to discover possible vulnerabilities, cryptographic misuses at a binary level, and other security weaknesses make it a powerful tool for identifying and mitigating security risks in IoT/embedded devices.

# Contents

# 1 Introduction

Devices connected to the Internet grow rapidly every year, especially devices that perform special purpose tasks, called Internet of Things (IoT)/embedded devices. In almost every field, applications of IoT technology offer multiple benefits. Home users, business corporations, and governments have started the adoption of IoT/embedded devices. Devices are used, from reliable Internet connections, smart sensors, and security systems to critical infrastructures such as power grids, hospitals, transportation systems, etc. For instance, routers and switches must provide a stable secure connection to the whole infrastructure, various sensors such as smoke gas detectors and current measurements must be reliable, as well as security camera systems such as CCTV. In recent years, router devices have supported a connection to the Internet infrastructure in order to provide the flexibility of controlling and monitoring them by distance. As more and more IoT/embedded devices gain access to the Internet (either directly or through an infrastructure) and the network of physical devices gets larger, new attack vectors rise, and devices may become susceptible to cyberattacks.

The rest of this introduction chapter starts with the given problem that outlines recent attacks on IoT/embedded devices, especially on the software (firmware or firmware image) level. Furthermore, research questions are given on how to effectively check, prevent and enhance the security of firmware images. Following the problem, this thesis tries to address a possible solution with an automated way of effectively checking the software security of IoT/embedded devices in order to minimize the overall security attack vectors and misuses that a developer may encounter. Lastly, the outline of the rest of this work is presented.

## 1.1. Problem Statement

The implications of the Internet of Things (IoT) and embedded devices are not limited to the cyber world but also extend into the physical world. In this context, cyber-attacks have the potential to cause direct physical harm[1,2,3,4]. Year by year, IoT/embedded devices attacks grow rapidly [24, 60]. Additionally, an increase is also observed in IoT malware [24]. Firstly, the *Persirai Trojan* that it could run arbitrary code execution on devices with super-user privileges in over $1,000$ different models of IP cameras[4]. Following with the *Mirai botnet* [28] that performed among the largest distributed denial-of-service (DDoS) attacks in 2016, using more than 600k infected IoT devices, where the implications of these attacks were to temporarily shut down massive networks including Internet Service Providers (ISPs), Government, Educational and Financial institutions. Researchers continue to successfully find vulnerable IoT devices, also in specific models or even at large scale [68]. IoT devices are also deployed in critical infrastructure such as a power grid structure and, unfortunately, can be susceptible to various attacks as

[1] "Ukraine power grid hack", Wikipedia, 2015, `https://en.wikipedia.org/wiki/2015_Ukraine_power_grid_hack`

[2] "Hackers Breach Thousands of Security Cameras, Exposing Tesla, Jails, Hospitals", Bloomberg, 2021, `https://www.bloomberg.com/news/articles/2021-03-09/hackers-expose-tesla-jails-in-breach-of-150-000-security-cams`

[3] "Smart camera and baby monitor warning given by UK's cyber-defender", BBC, 2020, `https://www.bbc.com/news/technology-51706631`

[4] "The Persirai Botnet", University of Hawaii-West Oahu, 2017, `https://westoahu.hawaii.edu/cyber/regional/gce-us-news/the-persirai-botnet/`

well [88]. News reports frequently highlight numerous attacks on IoT devices, ranging from critical infrastructure vulnerabilities to everyday personal use[1,2,3]. Overall, the security breaches of IoT devices have risen over the years, and more security measures need to be added as the consequences of these exploitable devices affect home users, business corporations, governments, and critical infrastructure.

An IoT/embedded device's software, also called firmware or firmware image, plays a significant role in the device's overall security. Previous studies on IoT devices [36, 97, 102] identified many software issues that an adversary can exploit and possibly gain access to the device. One of the main problems on these devices lies at the firmware level and the need for constant and frequent updating (Regular Security Updates). There are smart devices in which the firmware update comes directly from the vendors or manufacturers, or even, in the worst case, some devices do not have the mechanism to install a firmware update. The level of security in a firmware image depends on the development practices followed by the device manufacturer. Following secure coding practices, conducting thorough code reviews, and adhering to security guidelines can help mitigate vulnerabilities (Secure Development Practices). Devices may also come with passwords and/or private credentials that can be found embedded in the firmware image. Moreover, several factors can contribute to weakening a device's software security. A few of them are using outdated libraries with Common Vulnerabilities and Exposures (CVEs), the absence of binary exploit mitigation techniques, and API (Application Programming Interface) cryptographic misuses.

Generally, the main questions that this work will try to provide a solution concerning firmware image security are:

1. How secure is a firmware image of an IoT/embedded device in the context of secure development practices, vulnerability management, and regular security updates?
2. What changed/improved over the years in firmware image security?
3. How to effectively check the overall security of firmware images on a large-scale for different product types from various vendors?
4. Is there an automatic way to capture and limit common security mistakes in a firmware image even before a vendor releases it?

## 1.2. Proposed solution and contributions

In this work, an automatic security analysis tool for IoT/embedded firmware images is implemented and developed aim to discover possible weaknesses in a device's software. The automatic firmware security analysis tool is called ***E***mbedded ***B***inary ***A***nalysis ***T***ool (`EBAT`) and is provided open source[5]. IoT/embedded security analysis must be performed early, in the developed stages of a product's firmware image before any production release, in order to capture and inform the developer about the best practices for enhancing the overall device security. Most of the firmware images that are publicly available are closed source (black box). To address the previous questions regarding the security of a firmware image and even comparing multiple firmware images over multiple years, an automated tool is implemented in order to conduct a large-scale security context analysis at the firmware image level.

This automatic firmware analysis tool analyses the security of a firmware image in several stages and aims to find and inform the developer about possible security weaknesses before releasing a firmware image. `EBAT` also has the ability to analyze multiple firmware images at once, starting from the initial firmware release to the latest provided one. In that way, we can compare and provide results of how secure the firmware images of IoT devices have been over the years. A comparison can be made between several factors: comparing known libraries, percentage of updated binaries, common vulnerabilities and exposures mainly on cryptographic libraries and others. Moreover, API cryptographic misuses implemented on the binary level can also be discovered with a set of common cryptographic rules, as will explained in later chapters. Overall, the implemented tool is provided as open source, is versatile and is separated into modules in order for developers to add/modify a module to meet their specific requirements.

Our main contributions to this thesis are the following:

[5] `EBAT` is provided open source at `https://github.com/ppanagiotou/EBAT-public`

1. We developed ***E**mbedded **B**inary **A**nalysis **T**ool* (EBAT) an open source automatic tool that analyses cross-architectural firmware images security context. EBAT consists of various modules aimed to explore multiple security guidelines as follows:
   (a) Discover outdated software for various libraries focusing on cryptographic libraries.
   (b) Detect Common Vulnerabilities and Exposures (CVEs) focusing on firmware's cryptographic libraries binaries.
   (c) Detect exploit mitigation techniques on firmware's image binaries.
   (d) Credentials and passwords scanner mainly focuses on private keys embedded in the firmware image and the firmware's image binaries.
   (e) Comparing firmware image updates over the device's lifetime or at the latest available firmware image, along with simple binary level diffing analysis using fuzzy hashing.
   (f) Identify API cryptographic misuses using static taint analysis on cross-architectural binaries. We present a total of 18 well-defined cryptographic rules that are categorized by their cryptographic primitive and created a list of 733 function calls, along with their arguments, that can be used for checking the possibility of a cryptographic misuse based on 10 well-used open source cryptographic libraries APIs.
2. Using EBAT, we conducted a large-scale analysis on more than 36,000 firmware images publicly crawled from the Internet and successfully unpacked over 22,000. The created dataset of firmware images belongs to more than 5,000 different products across 33 vendors for a plethora of device types in a period of over 20 years. Our findings raise questions regarding the overall firmware image security of IoT/embedded devices.

## 1.3. Outline

The rest of this work is organized as follows:

**Chapter 2** covers the background material needed for understanding the rest of this thesis. Briefly explain what a firmware image is, what kind of firmware images exist and the security guidelines on IoT/embedded devices that are widely available. In Chapter's Section 2.3, the 18 cryptographic rules that are checked for cryptographic API misuses are presented and categorized by their cryptographic primitives.

**Chapter 3** presents prior research regarding the security of IoT/embedded devices focusing on a firmware image level.

**Chapter 4** contains a comprehensive explanation of the developed automated tool called EBAT. For each section in this chapter, a module is described in terms of how it is developed, its usage and what it can/cannot analyze.

**Chapter 5** presents our evaluation on a large scale of more than 36,000 firmware images belonging to more than 5,000 products harvested from 33 vendors (publicly available) in a period of over 20 years. Our implemented tool, EBAT, is executed on every product's firmware image, and all the results and findings are presented in this chapter. Additionally, we evaluate EBAT through case studies where researchers discovered vulnerabilities in firmware images and compared them with our automatic tool to verify and test the effectiveness of our implemented tool.

Finally, **Chapter 6** provides future improvements for EBAT and concludes the presented work.

# 2
# Background

This chapter includes the background material necessary for a better understanding of the rest of this thesis. Section 2.1 introduces firmware images and binaries in the Internet of Things(IoT)/embedded devices. In Section 2.2, the security guidelines that exist on embedded/IoT devices, binary analysis techniques and taint analysis are discussed. Additionally, an introduction to *Ghidra SRE* [70] is presented as the reverse engineering tool used in our code analysis implementation. Last but not least, the chosen cryptographic misuse rules categorized by their cryptographic primitive are mentioned and explained. These rules are applied at a software level to many standard cryptographic libraries on their well-defined Application Programming Interfaces (APIs).

## 2.1. Firmware images and binaries

A firmware image or firmware is the software used on embedded and IoT devices. Firmware is defined in [23] as "a combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device". It is delivered in various file formats, either standardized or not, and each vendor may use its own unique format variation. Each firmware image can contain one large single binary or a combination of multiple binaries along with other data and metadata. Linux-based firmware images are the most popular in embedded and IoT devices [36]. There are mainly two categories that are widely used in an embedded/IoT operating system (OS):

1. Embedded Linux OS, e.g., OpenWRT, Ubuntu, BusyBox.
2. Real-Time Operating Systems (RTOS), e.g., FreeRTOS, Mbed OS, Nucleus RTOS, QNX.

In this work, the term binary refers to machine code instructions (binary code) that are included within an executable file format. Various executable file formats exist, like Executable and Linkable Format (ELF), Portable Executable (PE) and binary raw format. The latter depends on each device's hardware (e.g., NVRAM capacity, processor architecture and peripherals) and contains specific headers that mainly declare the memory regions. An ELF binary is categorized as an executable, shared library, or an object file. Binaries use a variety of libraries that are dynamically and/or statically linked. Dynamic (shared) libraries are libraries that are shared in memory and can be used among many binaries. During the linking phase (a compilation phase of a binary), the addresses for the dynamic libraries are not known. Thus, references are not resolved, and the linker leaves *symbolic references* to these libraries until the binary is actually loaded into memory for execution. This is in contrast to static libraries that are merged into the binary executable, thus making the binary larger, but all references to them can be resolved directly.

Packing a firmware image is the method used to compress all the software elements (e.g., various executable or library binaries and data such as web files, configurations, etc.) together. Packing is mainly used for distribution, where unpacking is the inverse process in order to deploy ('flash' or 'burn') the firmware image into the device. No official standards are used for packing or unpacking; hence, each vendor uses its own method, algorithm or procedure for creating, updating, extracting and deploying the firmware image.

Overall, in order to analyze a firmware image and all its data, including binaries, the first step is to unpack the image. The main tools widely used for unpacking a firmware image are Binwalk [54], BANG [2], FACT extractor [46] and firmwalker [4]. When evaluating the aforementioned tools, a range of advantages and disadvantages become apparent. However, *obfuscation firmware techniques* exists that eventually will prevent any unpacking method from being successful. For instance, encrypting a firmware image prevents extraction without acquiring a decryption key.

## 2.2. Security Analysis on IoT/embedded devices

This section provides an overview of the security guidelines and binary analysis techniques employed in the research. Additionally, we briefly discuss the use of Ghidra SRE [70], a powerful software reverse engineering tool. Many guidelines on enhancing the security of embedded/IoT devices are reported in Open Web Application Security Project (OWASP), IoT Top 10 [81] and Embedded Application Security Best Practices [78], on both application and threat assessment level. Specific guidelines relevant to later chapters are presented and examined below.

Starting with the OWASP IoT Top 10 [81], the first guideline is to avoid the usage of "*Weak, Guessable, or Hardcoded Passwords*", meaning that a device should not keep its passwords in plaintext, nor easily accessible. The same applies to credentials, e.g., unencrypted private keys and SSH private keys. Passwords and credentials should preferably be stored in a hardware security element (SE) or in a Trusted Execution Environment (TEE). Whenever allowed by the device environment, a user may be able to change them. A hardware security element (SE) is a secure chip that offers multiple protections for tampering, resistance from side-channel and fault attacks, software attacks, etc. Thus, it usually offers confidentiality and integrity of the data that resides in the chip. A TEE is a secure area, mainly located as a part of the main processor (e.g., ARM TrustZone[1]), where it offers an isolated execution environment for executing code, detached from other parts including the remaining area of the main processor. Additionally, easily brute-forced or commonly used passwords should be avoided. Further guidelines refer to the "*Insecure Network Services*", where unneeded or insecure services running on the device itself should be disabled. For instance, the `'telnetd'` service is better to be inactive or even removed unless it is strictly needed. The "*Lack of Secure Update Mechanism*" guideline refers to unsecured firmware deployment capabilities to a device. Countermeasures include but are not limited to encrypted channels and signature verification mechanisms. The "*Use of Insecure or Outdated Components*" guideline includes deprecated or unsecured components/libraries whose usage should be avoided. The possibility of having an outdated library version on the developed tool-chain and the lack of checking and updating them results in the usage of these libraries across multiple firmware versions or even similar products. Last but not least, the "*Insecure Default Settings*" guideline refers to the insecure configuration of files such as web server configurations (e.g., `'lighttpd.conf'`) and even ssh daemon configurations (e.g., `'sshd_config'`), should be avoided by vendors. Additional measures, reported by [78], include the "*Usage of Debugging Code and Interfaces*" that it should be removed in a production release firmware while "*Transport Layer Security (TLS)*" may be utilized whenever is possible.

Security hardening features on binaries make use of mitigation techniques that increase the difficulty for an adversary to exploit the binary (also called exploit mitigation techniques). These features can reduce or even prevent buffer/stack/heap overflow attacks, Global Offset Table (GOT) attacks and many others. OWASP recommends C-based toolchain hardening in [79]. As an example, the GCC compiler has plenty of compiler flags/options that result in good hardened settings, such as Stack Smashing Protector (SSP) (also called a stack canary), Position Independent Code (PIC), Position Independent Executable (PIE) with Address Space Layout Randomization (ASLR), partial and full read-only allocations (RELRO). The developer should follow the recommended guidelines as presented in [79], which may harden or even prevent an adversary from creating a successful binary exploit. Furthermore, a developer may avoid (where possible) or limit the usage of known dangerous functions (e.g., strcpy) but instead make use of the safer alternatives (e.g., strncpy). In addition, the developer should ensure that all non-trusted data and user input is validated, sanitized, and/or output encoded to prevent unintended system executions [78]. Recent examples of injection attacks on embedded/IoT devices that are reported in *CVE-2020-15631* (OS command injection), *CVE-2020-8863* and *CVE-2020-8864* are possible due to incorrect handling of receiving data.

[1]"TrustZone for cortex-M", ARM, `https://www.arm.com/technologies/trustzone-for-cortex-m`

Recent binary exploitation attacks, particularly on embedded/IoT devices, were mainly possible due to the lack of binary exploit mitigation techniques. As an example, a recent attack presented in [101] with an exploitation illustrated on *ZDI-20-709*[2] allows an adversary to bypass authentication and execute code in the context of root. This attack was possible due to the lack of stack canaries and PIE on the '`httpd`' executable. The exploit could be hardened or even prevented if the aforementioned mitigation techniques existed. Similar attacks presented in *CVE-2020-10881* caused a stack-based buffer overflow due to crafted DNS messages (possibly mitigated by SSP), and in *CVE-2019-17147* caused an overwrite of GOT addresses by sending specific HTTP requests (possibly mitigated by RELRO). A remote attacker can use both exploits to execute arbitrary code.

Generally, the security analysis on a firmware image can be done manually, automatically or by a combination of both. Manual audit security analysis on a firmware image is extremely slow, time-consuming, and requires much human effort. Thus, there is a problem in terms of scalability when a large dataset of firmware images needs to be analyzed. However, it provides accurate results and yields findings that an automatic analysis would be challenging or even infeasible to discover. On the other hand, automatic security analysis is fast and scalable most of the time, but it often provides inaccurate or incomplete results. A combination of both is also possible, starting with an automatic analysis of a large dataset that may provide hints of possible weaknesses in a device, followed by a manual audit of the particular firmware to verify the weaknesses and/or discover additional ones.

### 2.2.1. Binary Analysis Techniques

Various binary analysis techniques/methods that are widely used, individually or combined, exist. In this section, disassembly and the benefits of using an intermediate representation (IR) form are presented. Additionally, various binary analysis techniques are briefly described, and taint analysis is introduced, together with the advantages and disadvantages of dynamic versus static taint analysis.

*Disassembly* is the process of translating machine language to assembly instructions. There are two main categories of disassembly: static and dynamic. The former attempts to extract the instructions of a binary without executing them. The latter, also called execution tracing, logs each executed instruction as the binary runs. When disassembling the machine instructions, the next logical step is to perform binary analysis techniques on the disassembled binary. However, the analysis techniques will only be available on a specific processor architecture. If we include binaries with different architectures, all analysis methods need to be rewritten, thus making the process time-consuming and sometimes imprecise. To overcome the previous problem, *intermediate representation/language (IR/IL)* forms are necessary, which are used for abstracting/translating the machine instructions from numerous architectures to a new unified language. This process is also called *lifting*. IR/IL form must be general enough to model the behaviour of different processors. Therefore, using IR, all the analysis techniques (algorithms and applications) can be developed on a common framework, reducing the complexity, time and effort.

Analysis techniques may consist of building the *Control Flow Graph (CFG)* and *Call Graph (CG)* of an executable. CFG represents the control flow in single-function basic blocks, where each basic block consists of sequences of instructions with no branches. The first instruction is the entry point, and the last one is the exit point. A call graph depicts the control flow of functions, in other words, the relationship between call sites and functions. In addition, every analysis may have one or more properties, with a number of them being: *inter-procedural* or *intra-procedural*, *context-sensitive* or *context-insensitive*. Briefly, *inter-procedural analysis* considers an entire program as a whole, typically by linking all the function's *CFGs* together via the call graph. Compare to *intra-procedural analysis* that considers only a single function at a time and thus analyzes the *CFG* of each function in turn. The *intra-procedural analysis* has the disadvantage of not being complete, meaning that it does not have the capability of combining different functions to obtain a result. The *context-sensitive* analysis considers the order of function invocations into account and computes a separate result for each possible path through the call graph. The accuracy of this analysis technique is bound by the call graph, which is limited by the completeness and accuracy of the call graph produced for a given program. On the other hand, *context-insensitive* analysis computes a single global result, which is typically faster. However, the results of a *context-insensitive analysis* may not be as accurate as those of a context-sensitive analysis, as it does not take into account how the program may behave differently in different contexts. Finally, *data-flow*

[2]"ZDI-20-709: Heap Overflow in the NETGEAR Nighthawk R6700 Router", Zero Day Initiative, 2020, `https://www.thezdi.com/blog/2020/6/24/zdi-20-709-heap-overflow-in-the-netgear-nighthawk-r6700-router`

*analysis* shows information about the data-flow. An example is *use-def chains*, which describes where a variable is used and defined at each point in the program analysis.

*Taint Analysis* is the process of tracking the data flow of selected data, called *taint*, to observe which program locations are affected. This can be done dynamically as the binary executes (*Dynamic Taint Analysis (DTA)*), or statically without executing it (*Static Taint Analysis (STA)*). In both cases, the taint sources, taint sinks and taint propagation need to be defined:

- *Taint sources* are the selected data, memory locations, registers, etc., that a user is interested in tracking. Thus, a user should define which data are marked as taint sources.
- *Taint propagation* is the process of propagating from taint sources to taint sinks. It operates on input operands of an instruction, acting on how it resolves to the output operand.
- *Taint sinks* are the endpoints that are influenced by the selected taint sources. For example, a user should define the endpoint when the taint propagation reaches an immediate value.

In the following paragraph, a comparison of dynamic taint analysis (*DTA*) versus static taint analysis (*STA*) is discussed through the context of applying them in IoT/embedded devices. In order to successfully perform a *DTA*, you need to have the platform/device on which the binaries can be executed. Otherwise, you need to perform a successful full system emulation, which is hard, time-consuming and sometimes imprecise due to the diversity of various devices' peripheral modules in IoT/embedded devices. For a general-purpose machine that has a *x86* or *x86-64* processor architecture, a *DTA* can be performed successfully on binaries compiled with identical CPU architecture using many tools, e.g., Intel PIN [62].

In this work, binaries from successfully unpacking public source firmware images are analyzed. The IoT/embedded firmware image binaries originate from various architectures, e.g., *x86, MIPS, ARM, PowerPC* including different endianness (e.g., little (*LE*) or big-endian (*BE*)), with numerous address sizes, e.g., 32 or 64 bit. Likewise, each firmware image corresponds to an embedded device that has plenty of peripherals depending on its usage, such as Non-Volatile Random-Access Memory (*NVRAM*), Electrically Erasable Programmable Read-Only Memory (*EEPROM*), General-Purpose Input/Output (*GPIO*), Web/Internet interfaces, various sensors, wireless chipsets and many other peripherals. A special case for many devices is the usage of their *NVRAM* or *EEPROM* in order to store bytes of data, meaningful for each device's functionality, that will persist without power. Those bytes of data may be unique per device, per firmware image, per product, and many times unknown. Usually, a device's physical acquisition is needed to perform a successful *DTA* on firmware's binaries. To reduce this gap, full system emulators like *QEMU* [15] exist. *QEMU* is an open-source emulator and virtualizer that can perform a full system emulation in many architectures. Thus, the problem is reduced to the successful system configuration of the emulator, e.g., *NVRAM, EEPROM* and peripherals configuration, which may or may not be possible in multiple cases due to the diversity of firmware images. A full system emulation performing *DTA* is by far more computationally expensive than *STA*, while the successful configuration of the emulator is often limited, thus making a successful *DTA* analysis not easily scalable.

*STA* is performed on intermediate representation *(IR)* or intermediate language *(IL)* forms of the binary's code. Thus, it supports analysis on multiple architectures (cross-architectural binary analysis), improving its versatility and enabling a more comprehensive understanding of the binary's behaviour. In these cases, it has the advantage of better scaling on larger datasets due to less computational requirements and specific configurations than *DTA*. Furthermore, *STA* can provide findings and useful insides for a device firmware image without the need for physical device acquisition and works on all successfully unpacked firmware images without the need of uniquely configuring each device. Lastly, a combination of *DTA* and *STA* techniques and probably the physical acquisition of a device is needed for the successful verification and exploitation of a discovered weakness/vulnerability.

There are techniques to keep the code secret and prevent reverse engineering on binaries, mainly deployed on malicious software (malware). These may also affect the binaries of unpacked firmware images (depending on the vendor); therefore, a subset of anti-reverse engineering techniques [39] are covered next. *Anti-Static analysis techniques* can target disassemblers in order to cause incorrect or partial disassembly called *disassembly desynchronization*. Additionally, these techniques may obfuscate the control flow, imported functions, or even opcodes. *Dynamically computed addresses* techniques, aimed to obfuscate the actual control flow path, cause the static analysis process to fail due to a complex or even infeasible way to compute the actual jump address. *Obfuscated Control Flow* techniques try to hide the control flow using multiple threads, child processes, or exception handlers for computing the actual control flow information. *Opcode Obfuscation* techniques intend to encode or encrypt machine instructions when the executable file is being generated. Thus, *opcode deobfuscation* must be performed before actual execution. *Imported Function Obfuscation* focus on hiding which dynamically linked libraries and their corresponding functions are being used for the purpose of avoiding any leaking information. *Anti-Dynamic Analysis* techniques also exist and aim to prevent dynamic analyses on a binary. These techniques will not be discussed further, as the rest of this thesis focuses mainly on static analysis techniques.

### 2.2.2. *Ghidra SRE*

*Ghidra* is an open-source software reverse engineering (SRE) tool suite developed by the National Security Agency (NSA) [70]. *Ghidra* can support a plethora of instruction set architectures (ISA), for instance, x86 16/32/64 bit, ARM and AARCH64, PowerPC 32/64 bit and MIPS 16/32/64 bit. As an open-source tool, the best advantage of *Ghidra* is that it allows one to develop scripts/plugins and share them with the community. It offers a Graphical User Interface (GUI) and headless scripts for non-user interactions for automating repetitive tasks. In addition, many analysis features and techniques are already developed in order to enhance the analysis of a binary.

*Ghidra* analysis consists of various analysis tools called analyzer tools/plugins, such as function, stack, cross-reference, entry point and demangle analyzer, that can be activated either manually or automatically. Additionally, it consists of analysis watches that monitor and act on specific changes. For instance, a disassembly watch constantly monitors for new disassembled chunks of memory and triggers relevant analyzer plugins automatically. *Ghidra* starts at entry points and disassembles the memory by following flows. When a new memory area is disassembled, multiple analyzers can be initiated, either prioritized by the disassembly watch or run in parallel to analyze specific changes. The priorities play a significant role as, for example, a *Stack analyzer* can not start before a *Function analyzer* as no new function has been discovered yet. The analyzers briefly discussed below are only a limited subset of what *Ghidra* offers. A *Function analyzer* is responsible for creating any new functions and/or function calls if the new disassembled memory corresponds to the start of a new function's basic block. A *Cross-reference analyzer* will create the references between those function calls, while a *Stack analyzer* tries to build a stack based on any discovered stack references. A *Data reference analyzer* looks at references for possible strings or pointers to code, and an *Entry point analyzer* disassembles code at starting symbols/addresses and marks them as external entry points. Last but not least, a *Demangler analyzer* is responsible for taking mangled symbol names generated by compiling object-oriented language code, e.g., C++, and converting them back into their original, human-readable form. Overall, *Ghidra* analysis improves and expands with every new public release of the tool.

A binary can be imported into *Ghidra* using the GUI or using the headless mode that requires no user interaction. Headless scripts have the same capabilities as the GUI. However, they offer enormous flexibility when performing repetitive tasks on numerous binaries. After importing, a user can select the analysis options, a number of them mentioned above. Initially, *Ghidra* tries to disassemble the binary in order to extract the assembly instructions from it. The supported architectures are specified by *SLEIGH*, which is a language for describing the instruction sets of general-purpose microprocessors. Also, it specifies the translation from a machine instruction to *P-Code* (IR form). If a processor is not supported by *Ghidra*, a user can add it using the *SLEIGH* language. *P-Code*, from *Ghidra*'s documentation, is a Register Transfer Language (RTL), distinct from *SLEIGH* and designed to specify the semantics of machine instructions. *RTL* is a class of IR/IL forms. After disassembly, *Ghidra* will eventually lift the binary to *P-Code*. Many of the analysis techniques that *Ghidra* can perform are using *P-Code's* IR form. This includes the static taint analysis headless scripts that we developed and used in this thesis, which have the advantage of analyzing multiple binaries from various CPU architectures using the same headless developed scripts.

In the rest of this section, a brief overview of *P-Code* internals is provided to help understand the functioning of our developed headless scripts. For a more detailed understanding, please refer to the Ghidra documentation [70]. As previously mentioned, *P-Code* is an *RTL* form generated by *SLEIGH* language. The process of converting processor instructions into a series of P-code operations, called lifting, involves using parts of the processor state as inputs and outputs, known as varnodes (which will be explained later). This direct translation of instructions is referred to as raw *P-Code*. Each raw *P-Code* operation can directly emulate an instruction execution. The creation of raw P-code is a crucial step in constructing a graph, but further steps are necessary, including the addition of pseudo operations, such as `MULTIEQUAL` and `INDIRECT`, which are new opcodes that do not directly emulate an instruction. Instead, these pseudo operations emulate a set of instructions, not a single instruction.

A *P-Code* operation is the analogue of an assembly instruction operation, e.g., addition, store, move, etc., where the action is determined by its opcode. Overall, the basic format of a *P-Code* operation consists of one or more input varnodes and optionally produces a single output varnode. Indirect effects are only possible in pseudo operations. For all other *P-Code* operations, only the output varnode can have its value modified. A *varnode* explained at *Ghidra*'s documentation as: "A varnode is a generalization of either a register or a memory location. It is represented by the formal triple of an address space, an offset into the space, and a size. Intuitively, a varnode is a contiguous sequence of bytes in some address space that can be treated as a single value. All manipulation of data by *P-Code* operations occurs on varnodes." An address space is a generalization of RAM, which may consist of a *ram* space, a *register space*, a *constant* space or a *temporary* space. Briefly, for a typical processor, *ram* space is used to model memory accessible via its main data bus, *register* space is used for modelling the processor's general purpose registers, *constant* address space is used to encode any constant values needed for *P-Code* operations, and lastly, *temporary* space is used to model temporary registers that may use to hold intermediate values when modelling instruction behaviour.

## 2.3. Cryptographic Misuse Rules

Standard cryptographic libraries contain well-implemented and well-defined application programming interfaces (APIs) that a developer can use to implement cryptographic features in a device's software (firmware). However, a developer might not use the API correctly, potentially compromising the intended security function, either by using deprecated function calls or by applying improper function arguments (security issues). In this section, rules for cryptographic misuse of commonly used cryptography primitives are described. Those cryptographic misuses correspond to the developers' improper usage of the cryptographic APIs, which may lead to a potential security issue. The cryptographic misuse rules are explained in subsequent sections and are created based, but not limited to, the following studies, guidelines and references: OWASP Testing for Weak Encryption guideline [80], OWASP IoT top 10 [81], Zhang et al. CRYPTOREX [102], Egele et al. [40], Lazar et al. [55], RFC 2313 [18], RFC 2437 [19], RFC 8017 [21], NIST 800-131A [71] and NIST 800-132 [72] as well as other NIST publications. It should be noted that more cryptographic misuse rules can exist, and the following is a limited subset used in our framework.

The presented work focuses on standard cryptographic libraries, e.g., OpenSSL, GnuPG, mbedTLS, WolfSSL, etc., that are dynamically linked on a binary. Statically linked cryptographic libraries and firmware images that use their own cryptographic implementations are left for future work. All covered cryptographic libraries are presented in Chapter 4.6, and for each library, we *assume* that cryptographic primitives in the listing are implemented correctly and *securely*, both on an algorithmic and application level. Attacks such as side-channel (cache, timing, power, etc.) are out of the scope of this work, as well as attacks like buffer, stack or heap overflows, e.g., heartbleed. Every presented cryptographic library has a well-defined application programming interface (API). Our work focuses on the *improper usage of cryptographic function calls and their corresponding arguments* by the firmware developer using the defined API. Inappropriate use (security-wise) of such functions and/or functions arguments may originate due to the product's lack of security by design or even misconceptions/misunderstandings by the developer implementing the security/cryptography of a device using the well-defined API. Over time, this may compromise the overall device security, as many real-world examples have shown.

For every cryptographic misuse rule, a class of functions, function arguments and misuse conditions are defined. For instance, the misuse rule $R1$ class consists of several functions with several specific function arguments and multiple misuse conditions. Each misuse condition is an expression resulting in true or false. The rule is violated if the misuse condition is triggered (true). For example, consider the following function prototype '`void encrypt(const char *key)`', in which the function '`encrypt`' and function argument '`key`' belong to the misuse rule $R1$ class. The misuse condition expression is defined as '`if(key == constant value) then true else false`', where if the key is found to be constant/fixed, then the rule is violated; otherwise, it is not. Those classes are found based on the device's firmware code and a tremendous manual effort to find the API's calls for chosen standard cryptographic libraries. Additional information about this topic can be found in Chapter 4.11.

The cryptographic functions and their corresponding arguments, and the misuse conditions which belong to each cryptographic misuse rule class, are presented in the implemented open-source tool given in Chapter 4 and explained in more detail in Chapter 4.11. To discover the value of a function's argument, code analysis (static taint analysis) is implemented; more details are provided in Chapter 4.12. Be aware that not all arguments are needed for a misuse condition. However, some arguments may provide useful insides and metadata, for instance, an argument that defines the length of a key.

In the following subsections, where each one covers one cryptographic primitive, the cryptographic misuse rules are presented with a brief explanation for each rule. Additionally, for each subsection, a table is presented, including **examples** of such cryptographic misuses in defining the function and function argument of interest, along with the misuse condition where it will be violated if the expression holds true. Each row of the example tables presented in subsequent sections consists of six columns, where the last two are optional. The 1$^{st}$ column shows the rule that refers to, and the 2$^{nd}$ column shows the function prototype of our function of interest and under which library it belongs. The 3$^{rd}$ column presents the argument that is considered for a cryptographic rule violation, and the 4$^{th}$ column depicts the misuse condition that if it is found to be true, then the rule is marked as violated. The 5$^{th}$ column represents the metadata needed to resolve the misuse argument, and finally, the 6$^{th}$ column gives the metadata argument. The 5$^{th}$ and 6$^{th}$ columns may not be present in some tables or omitted if no metadata is needed to trigger the misuse condition. It should be pointed out that these examples are only a limited subset of the overall cover functions, functions arguments, and cryptographic libraries, and the complete list of them is provided as open source in our implemented tool code.

### 2.3.1. Symmetric Key Cryptography

The cryptographic misuse rules chosen and implemented for this work on Symmetric Key Cryptography primitives are presented below. Table 2.1 depicts a few **examples** of such cryptographic misuses for every rule.

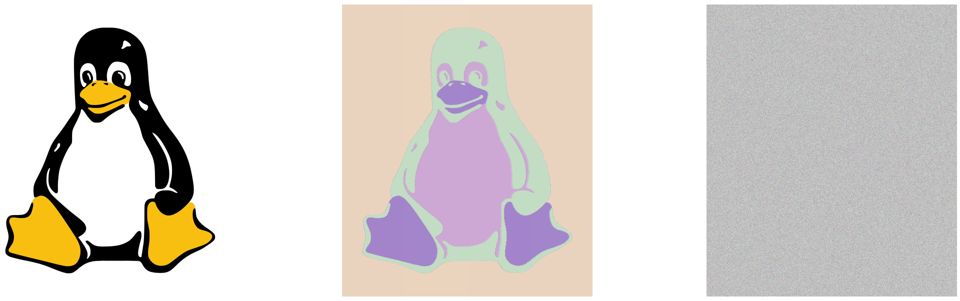
- **Rule S1:** *Usage of constant encryption/decryption keys for various block and stream ciphers.* The symmetric key should not be declared constant in a binary's data segment/section[3]. Additionally, it should be protected using a secure element when possible or generated dynamically, for instance, with a *CSPRNG*. An adversary can easily recover all ciphertexts by finding the symmetric key, thus breaking the encryption. It should be pointed out that a constant symmetric key hard-coded in a program's code (memory) is publicly available information and not a secret.

- **Rule S2:** *Usage of electronic code book (ECB) mode of operation ($>1$ block).* The *ECB* mode of operation on block ciphers has the weakness of discovering identical ciphertext blocks when encrypting identical plaintext blocks since blocks are encrypted independently from one to each other. Thus, *ECB* mode is deterministic and not Indistinguishably Under Chosen-Plaintext Attack (*IND-CPA*) secure. For instance, Figure 2.1 represents an image encryption using different modes of operation. Figure 2.1a depicts the original Tux image[4]. The image shown in Figure 2.1b is the encryption of the original image with *AES* [75], a key size of 128 bits (`0x000...00075BCD15`) and mode of operation *ECB*. The image is still visible despite the strong encryption that *AES* offers. Lastly, Figure 2.1c represents the encryption of the original image using *AES* with a key size of 128 bits, same as before, and IV equal to zeroes, using Cipher Block Chaining (*CBC*) mode of operation. The encrypted image is not visible using *CBC* due to the chaining mechanism that causes each new encrypted block to be dependent on all preceding blocks and the IV.

- **Rule S3:** *Initialization Vector (IV)/nonce repetition (fixed) on various modes of operation.* Encrypting a plaintext with an identical IV/nonce (e.g., in *CBC*, *CTR* etc.) will result in an identical ciphertext. The IV/nonce should be truly random to be *IND-CPA* secure. Encrypting with different IVs using the same key prevents the leakage of any information on ciphertext (i.e., non-deterministic). An attack reported on [64] for SSL version 3.0 and TLS version 1.0 using CBC mode of operation illustrates the necessity of truly random IVs.

[3] The data segment of an ELF binary contains sections such as '.rodata' (read-only data) section, which is dedicated to storing constant values that are not writable and '.data' section which may also have initialization/constant values used by variables but with writable permissions which means that it could be possibly changed across binary's execution.

[4] Tux, as originally drawn as a raster image by Larry Ewing in 1996 (Tux (mascot)).



(a) (b) (c)

Figure 2.1: (a) The original Tux image[4] (b) Tux image encrypted with AES, 128 bits, ECB (c) Tux image encrypted with AES, 128 bits, CBC

- **Rule S4:** *Usage of "weak" ciphers for encryption/decryption.* The ciphers that NIST 800-131A [71] declared as weak (Insufficient security strength) and disallowed are: *DES*, *Two-key TDEA (3-DES with two different keys), SKIPJACK, IDEA*. It is advised to avoid the use of *RC2, RC4 and Blowfish* as they have been shown to have insufficient security strength due to their small key size in various reported attacks [20, 47] (but not limited to). Decryption on those ciphers may be used only for legacy devices.

| Rule # | [Library]:Function prototype | Arg. | Misuse Condition | Metadata | Arg. |
|---|---|---|---|---|---|
| S1 | [OpenSSL]:<br>`int AES_set_decrypt_key(`<br>`const unsigned char *userKey,`<br>`const int bits, AES_KEY *key);` | 1 | constant bytes | size of key in bits | 2 |
| S2 | [OpenSSL]:<br>`int EVP_EncryptInit(`<br>`EVP_CIPHER_CTX *ctx,`<br>`const EVP_CIPHER *type,`<br>`unsigned char *key,`<br>`unsigned char *iv);` | 2 | context immediate value<br>`EVP_aes_128_ecb()` or<br>`EVP_aes_192_ecb()` or<br>`EVP_aes_256_ecb()` or<br>others | - | - |
| S2 | [GnuPG (libgcrypt)]:<br>`gcry_error_t gcry_cipher_open(`<br>`gcry_cipher_hd_t *hd,`<br>`int algo, int mode,`<br>`unsigned int flags)` | 3 | integer immediate value<br>`GCRY_CIPHER_MODE_ECB = 1` | - | - |
| S3 | [GnuPG (libgcrypt)]:<br>`gcry_error_t gcry_cipher_setiv(`<br>`gcry_cipher_hd_t h,`<br>`const void *k, size_t l);` | 2 | constant bytes | size of key in bytes | 3 |
| S4 | [WolfSSL]:<br>`int wc_Des_CbcEncryptWithKey(`<br>`byte *out, const byte *in,`<br>`word32 sz, const byte *key`<br>`const byte *iv);` | - | usage of DES cipher for encryption | - | - |

Table 2.1: **Examples** for Symmetric Key Cryptography of Cryptographic Misuse Rules

An example follows to further illustrate a violation of rule *S1* that triggers when the symmetric key is found constant in an executable. Consider the following code snippet 2.1 that displays a violation of rule *S1* with a constant key of size 128 (0x80) bits long. The function we are checking for a violation is called `AES_set_decrypt_key()` depicted in line 3. This function sets a decryption key with the underlying algorithm to be *AES*. In order to search if the rule is violated, firstly, the size of the key needs to be determined. Thus, the second argument of `AES_set_decrypt_key()` needs to be resolved first that contains the key size in bits (metadata), which in this particular case is 16 bytes long. Afterwards, the constant key is resolved as '`{0x3, 0x4, 0x5, 0x6, 0x07, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12}`' (as it is saved in this case by the compiler in '.rodata' section), thus, the rule is marked as violated.

```
1 void wrapper_AES(uint8_t *param_1, int param_2){
2   AES_KEY *key; // output key stack variable
3   AES_set_decrypt_key(param_1,param_2,&key);
4   /* code */
```

```
  ...
}

uint8_t constant_key[] = {0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x07, 0x8, 0x9, 0xA, 0xB, 0xC,
              0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15};

int main(int argc, char **argv){
  // call of AES sets the decrypt key param_1 of size of parma2 = 128 bits
  // cryptographic misuse of rule S1
  wrapper_AES((&constant_key[0]) + 2, 0x80);
  /* code */
  ...

  return 0;
}
```

Listing 2.1: Code example of a cryptographic misuse of rule S1

### 2.3.2. Cryptographic hash functions

Cryptographic hash functions are often combined with other cryptographic primitives, i.e., digital signatures, Message Authentication Codes (MACs), Key Derivation Functions (KDFs) and Password Based Encryption (PBE); therefore, no cryptographic misuse rules can be directly defined. Generally, the MD family is deprecated since MD-5 is not collision resistant, as many attacks have shown [91, 96], making the hash function not secure for digital signatures and many other cryptographic applications. The first collision for full SHA-1 was reported by M. Stevens et al. [92] in 2017, and recent attacks such as [58, 59] render the usage for SHA-1 limited. SHA-1 is advisable to be avoided in digital signatures, and whenever it is feasible, applications may limit the usage of SHA-1 with other cryptographic primitives as well.

### 2.3.3. Public Key Cryptography

The cryptographic misuse rules chosen and implemented for this work on Public Key Cryptography primitive are presented below. Table 2.2 illustrates **examples** of such cryptographic misuses for certain open-source cryptographic libraries.

- **Rule P1:** *Usage of insecure RSA encryption padding schemes*. If no padding is used, also called “textbook RSA”, then the encryption scheme is *malleable* and *deterministic*, hence not IND-CPA secure. Public-Key Cryptography Standards (PKCS) #1 v1.5 padding introduced in RFC 2313 [18] adds redundancy to make the encrypted message *non-deterministic* together with additional checks against malicious message modifications. However, an adaptive chosen ciphertext attack was first reported by D. Bleichenbacher [31] with a proof of concept in *SSL V3.0*. Therefore, the PKCS #1 v1.5 padding must be avoided, and it is **not recommended for new applications** as stated on the latest PKCS #1 v2.2, RFC 8017 [21]. Optimal Asymmetric Encryption Padding (OAEP), first reported in PKCS #1 v2.0, RFC 2437 [19] is IND-CPA secure and is the recommended encryption padding scheme for every new application (CWE-780).

- **Rule P2:** *Digital Signatures signing/verifying with a “weak” cryptographic hash function.* NIST 800-131A [71] defines the approved hash functions families for signature generation and verification. The MD family (e.g., MD5) is deprecated, and SHA-1 is not recommended for generating a digital signature. SHA-1 may be used only for verification on legacy devices.

- **Rule P3:** *X.509 certificates signing/verifying with a “weak” cryptographic hash function.* MD family should not be used for any operation on certificates [57] [29], and SHA-1 may only be used for verification on legacy devices.

### 2.3.4. Pseudo Random Number Generators (PRNGs)

The cryptographic misuse rules chosen and implemented for this work on Pseudo Random Number Generators primitive are presented below. Table 2.3 represents **examples** of such cryptographic misuses.

| Rule # | [Library]:Function prototype | Arg. | Misuse Condition |
|---|---|---|---|
| Rule P1 | [OpenSSL]: `int RSA_public_encrypt(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding);` | 5 | integer immediate value `RSA_PKCS1_PADDING = 1` |
| Rule P1 | [mbedTLS]: `void mbedtls_rsa_set_padding( mbedtls_rsa_context *ctx, int padding, int hash_id );` | 2 | integer immediate value `MBEDTLS_RSA_PKCS_V15 = 0` |
| Rule P2 | [OpenSSL]: `int EVP_DigestSignInit( EVP_MD_CTX *ctx, EVP_PKEY_CTX **pctx, const EVP_MD *type, ENGINE *e, EVP_PKEY *pkey);` | 3 | context immediate value `EVP_md5();` or `EVP_md4();` or others |
| Rule P3 | [OpenSSL]: `int X509_digest( const X509 *data, const EVP_MD *type, unsigned char *md, unsigned int *len);` | 2 | context immediate value `EVP_md5();` or `EVP_md4();` or others |

Table 2.2: **Examples** for Public Key Cryptography of Cryptographic Misuse Rules

- **Rule R1:** *Usage of static seeds for pseudo random number generators (PRNGs) or Cryptographically-secure pseudorandom number generator (CSPRNGs) in a security context.* When a static seed is used, a *PRNG* will produce identical random number sequences each time. Thus, an adversary can 'guess' the next random number, reproducing the sequence. Hence, for cryptographic applications that request random numbers, e.g., the Diffie Hellman Key Exchange (DHKE), the security is compromised even if the underlying algorithm is secure.

- **Rule R2:** *Usage of low entropy sources for seeds on PRNGs or CSPRNGs in a security context.* Low entropy sources like predictable sources such as time (e.g., library libc, `srand(time())`) or process id (e.g., library libc, `srand(getpid())`) or any combination of them (e.g., library libc, `srand(time() + getpid())`), should be avoided for cryptographic applications, as presented in *CWE-337* and *CWE-338*. *CSPRNGs* must be seeded from '`/dev/random`' or '`/dev/urandom`' (on a Linux system) that provides securely unpredictable random bits.

| Rule # | [Library]:Function prototype | Arg. | Misuse Condition |
|---|---|---|---|
| Rule R1 | [glibc]: `void srand(unsigned int seed);` | 1 | integer immediate value |
| Rule R1 | [OpenSSL]: `void RAND_seed( const void *buf, int num);` | 1 | constant string |
| Rule R2 | [glibc]: `void srand(unsigned int seed);` | 1 | return value of `time()` or `getpid()` or any combination of them |

Table 2.3: **Examples** for Pseudo Random Number Generators of Cryptographic Misuse Rules

### 2.3.5. Key Derivation Functions (KDFs) and Password Based Encryption (PBE)

The cryptographic misuse rules chosen and implemented for this work on Key Derivation Functions (KDFs) and Password-Based Encryption (PBE) algorithms are presented below. Table 2.4 represents a few **examples** of such cryptographic misuses.

- **Rule K1:** *Usage of constant passwords/keys for PBE/KDFs.* Passwords should not be constant in a program's memory[3] and must be protected with a secure element, as it can be discovered with static code analysis. Additionally, it should be protected whenever is feasible with a secure element or dynamically generated, for instance, with a CSPRNG and stored in a hashed form along with a unique salt (not in plaintext).

- **Rule K2:** *Usage of constant salts, or no salts for PBE/KDFs.* Salt must also be provided and be unique/random for each password. Using a random/unique salt makes it difficult for an adversary to perform a precomputed dictionary-based attack, such as rainbow tables. In addition, two identical passwords are hashed differently using a unique salt and identical when using no salt or even the same salt. Hence, for each password, salt must be unique in order to have no collisions between identical passwords/keys.

- **Rule K3:** *Usage of "low" number of iterations in a KDF.* The total number of iterations that are considered "low" depends on the KDF algorithm and application level tolerance. A higher number of iterations makes it harder for an adversary to find the secret. According to NIST 800-132 [72], "the number of iterations should be set as high as can be tolerated by the environment, while maintaining acceptable performance." RFC 8018 (PKCS #5: Password-Based Cryptography Specification Version 2.1) [22] states that the minimum number of iterations should be set to $1,000$ while in OWASP [80] is recommended to be over $10,000$.

- **Rule K4:** *Usage of "weak" cryptographic hash functions or "weak" block ciphers for KDFs and Password Based Encryption (PBE) algorithms.* "Weak" hash functions, e.g., The *MD family*, and "weak" block ciphers, e.g., *DES* are recommended to be avoided by NIST 800-131A [71] and NIST 800-132 [72] in any new applications.

| Rule # | [Library]:Function prototype | Arg. | Misuse Condition | Metadata | Arg. |
|---|---|---|---|---|---|
| Rule K1 | [libcrypto]: `char* crypt(char* key, char* salt);` | 1 | constant string | - | - |
| Rule K2 | [wolfSSL]: `int wc_HKDF (int type, const byte *inKey, word32 inKeySz, const byte *salt, word32 saltSz, const byte *info, word32 infoSz, byte *out, word32 outSz);` | 4 | constant bytes | size of salt in bytes | 5 |
| Rule K3 | [OpenSSL]: `int PKCS5_PBKDF2_HMAC_SHA1( const char *pass, int passlen, const unsigned char *salt, int saltlen, int iter, int keylen, unsigned char *out);` | 5 | integer value weak iteration: $< 1000$ (RFC 8018[22]) | - | - |
| Rule K4 | [GnuPG]: `gpg_error_t gcry_kdf_derive( const void *passphrase, size_t passphraselen, int algo, int subalgo, const void *salt, size_t saltlen, unsigned long iterations, size_t keysize, void *keybuffer );` | 4 | integer immediate value `GCRY_MD_MD5 = 1` or others | - | - |
| Rule K4 | [OpenSSL]: `int EVP_BytesToKey( const EVP_CIPHER *type, const EVP_MD *md, const unsigned char *salt, const unsigned char *data, int datal, int count, unsigned char *key, unsigned char *iv);` | 2 | context immediate value `EVP_md5();` or `EVP_md4();` or others | - | - |

Table 2.4: **Examples** for Key Derivation Functions (KDFs) and Password Based Encryption (PBE) of Cryptographic Misuse Rules

### 2.3.6. Message Authentication Codes (MACs)

The cryptographic misuse rules chosen and implemented for this work on Message Authentication Codes (MACs) are presented below. Table 2.5 depicts **examples** of such cryptographic misuses, one for every rule for various cryptographic libraries.

- **Rule M1:** *Usage of constant/fixed authentication keys.* The key should not be declared in a binary's data segment/section memory[3] and should be protected with a secure element or dynamically generated, for instance, with a secure key exchange method. When finding the key, an adversary can easily recover and tamper the messages, thus breaking the authentication mechanism.

- **Rule M2:** *Usage of "weak" underlying cryptographic hash function for MACs. HMAC* as recommended by NIST 800-131A [71] may use any approved cryptographic hash functions. The *MD family* e.g., *MD5* hash function is recommended to be avoided.

- **Rule M3:** *Usage of insecure key lengths for MACs. HMAC* as recommended by NIST 800-131A [71] may use key with length greater than or equal to $112$ bits ($14$ bytes).

| Rule # | [Library]:Function prototype | Arg. | Misuse Condition | Metadata | Arg. |
|---|---|---|---|---|---|
| Rule M1 | [GnuPG]: `gcry_error_t gcry_md_setkey( gcry_md_hd_t h, const void *key, size_t keylen);` | 2 | constant bytes | size of key in bytes | 3 |
| Rule M2 | [OpenSSL]: `unsigned char *HMAC( const EVP_MD *evp_md, const void *key, int key_len, const unsigned char *d, int n, unsigned char *md, unsigned int *md_len);` | 1 | context immediate value `EVP_md5();` `EVP_md4();` or others | - | - |
| Rule M3 | [mbedTLS]: `int mbedtls_md_hmac_starts( mbedtls_md_context_t *ctx, const unsigned char *key, size_t keylen);` | 3 | $< 112$ bits | - | - |

Table 2.5: **Examples** for Message Authentication Codes of Cryptographic Misuse Rules

### 2.3.7. Authenticated encryption/decryption and AEAD

The cryptographic misuse rules chosen and implemented for this work on authenticated encryption/decryption schemes and authenticated encryption with associated data (AEAD) are presented below. Table 2.6 depicts **examples** of such cryptographic misuses, one for every rule, for various cryptographic libraries.

| Rule # | [Library]:Function prototype | Arg. | Misuse Condition | Metadata | Arg. |
|---|---|---|---|---|---|
| Rule A1 | [mbedTLS]: `int mbedtls_ccm_setkey( mbedtls_ccm_context *ctx, mbedtls_cipher_id_t cipher, const unsigned char *key, unsigned int keybits);` | 3 | constant bytes | key size in bits | 4 |
| Rule A2 | [WolfSSL]: `int wc_AesGcmEncrypt(Aes* aes, byte* out, const byte* in, word32 sz, const byte* iv, word32 ivSz, byte* authTag, word32 authTagSz, const byte* authIn, word32 authInSz);` | 5 | constant bytes | key size in bytes | 6 |

Table 2.6: **Examples** for Authenticated encryption/decryption and AEAD of Cryptographic Misuse Rules

- **Rule A1:** *Usage of constant/fixed encryption/decryption keys for various modes of operation of authenticated encryption/decryption and AEAD.* The authenticated key should not be declared constant in a binary's data segment/section memory[3]. Additionally, it should be protected whenever it is feasible with a secure element or dynamically generated, for instance, with a secure key exchange method. By finding the authenticated encryption key, an adversary can easily recover and/or tamper messages, thus breaking the authenticated encryption. It should be pointed out that a constant authenticated key hard-coded in a program's code (memory) is publicly available information and not a secret.

- **Rule A2:** *Initialization Vector (IV)/nonce repetition (fixed) on various modes of operation for authenticated encryption/decryption and AEAD.* For the GCM mode of operation, the IV can be created using a synthetic initialization vector construction, such as deterministic and RGB construction, where all must fulfil the requirement of "uniqueness". Using a constant IV, or even if one IV is ever repeated, then the implementation may become vulnerable to forgery attacks [52].

# 3 Related Work

In this chapter, prior research on the security of embedded/IoT devices is presented. There are numerous security analysis reports, including individual or on a large scale on embedded/IoT devices, as presented in section 3.1. These include static and dynamic analysis techniques on firmware images that aim to discover vulnerabilities and weaknesses. Section 3.2 presents prior studies for cryptographic function misuses for multiple platforms, including embedded/IoT firmware images. The authors of these studies aim to discover cryptographic implementation mistakes using code analysis techniques like static taint analysis.

## 3.1. Security Analysis of Firmware

Many researchers are interested in the security of embedded/IoT devices, as shown by the numerous studies performed on all levels over the years. The first public, large-scale analysis on embedded firmware images was presented in 2014 by Costin et al. [36], in which they performed a static analysis (not static code analysis) using a correlation engine to compare and find similarities between the captured objects on 32 thousand firmware images. The analysis discovered a total of 38 previously unknown vulnerabilities in over 693 firmware images that correspond to over 123 different products and affect at least 140K devices accessible over the Internet (2014). The aforementioned work managed to extract RSA keys and their self-signed certificates and hard-coded password hashes, most of them weak, and therefore recovered the original passwords. Additionally, the study discovered possible backdoors such as the authorized keys files (SSH keys), hard-coded web login admin credentials and hard-coded telnetd credentials.

A recent security report on home routers is presented by Weidenbach et al. [97] (2020). The study analyzed statically 117 firmware images without performing any code analysis and finding useful insides about exploit mitigation techniques, hard-coded credentials, private keys, and operating system versions that lead to critical known vulnerabilities. The study concludes that the old Linux kernel is still in use, and exploit mitigation techniques are nearly enabled on firmware's binaries.

Over the years, dynamic and static analysis techniques have also increased. In Chapter 2, Section 2.2, the advantages and disadvantages of those are explained. Avatar framework presented in [100] supports dynamic analysis on an embedded firmware with high accuracy of findings. However, it requires a physical acquisition of a device; thus, it cannot scale well. On the other hand, the dynamic analysis reported in [34, 37, 90] are scalable, although there are limitations in the analysis and accuracy due to specific hardware that an embedded device has. Dynamic analysis needs to solve the challenge of embedded systems specific hardware emulation, for instance, lack of *NVRAM* specific parameters, *init* and *rc* initialization scripts do not exist, or they are not trivial. Specifically, a dynamic analysis on embedded web interfaces (e.g., a firmware that embeds a web server) reported on [37] used the *QEMU* [15] full system emulator. The analysis scales relatively well as they evaluated a total of 1925 unpacked firmware images and were able to discover 225 previously unknown vulnerabilities in 45 firmware images. Additionally, they discover vulnerabilities such as SQL injection, command execution, XSS and CSRF. D. D Chen et al. introduced FIRMADYNE [34], which also performs automatic dynamic analysis using QEMU and tests known exploits on the firmware images. They found 14 previously unknown vulnerabilities for 69 firmwares and concluded that code-sharing is prevalent between manufacturers. FirmFuzz, presented on [90], is an automated framework using QEMU emulator that performs fuzz testing of vendor-developed applications on Linux-based embedded firmwares in order to find deep vulnerabilities. This framework ran on 6,427 firmware images and discovered 7 previously unknown vulnerabilities that affect 32 images, which corresponds to 6 devices in a total of 2 IP cameras and 4 routers. FIoT (Fuzzer of IoT) reported on [103] detected memory corruption using a combination of static and dynamic analysis (symbolic execution) for lightweight IoT firmware images and managed to discover 35 zero-day memory vulnerabilities among 115 firmware images.

A major challenge for code analysis on multiple firmware images lies in the different computer architectures, e.g., MIPS, ARM, PowerPC, etc. Cross-architectural bug search studies performed on binaries and firmware images reported over the years [42, 44, 82, 98] limit the gap of performing a large-scale code analysis between various architectures using existing tools. Particularly, *Genius* [44], a bug search engine that relies on the CFGs of binaries (extracted using IDA Pro [49]), reported potentially vulnerable firmware and confirmed some of them using a large dataset of IoT devices in a reasonable amount of time. A newer study called *Gemini* [98] used binary code similarity detection with deep neural network-based graph and identified more vulnerable firmware images than *Genius*, with better accuracy. Karonte [84] presented by N.Redini et al. performs a multi-binary static analysis on firmware images to identify insecure interaction between the binaries through a finite set of Inter-Process Communication (IPC) that may lead to vulnerabilities, e.g., buffer overflow, input data sanitization etc. The study discovered 46 zero-day bugs, examined 53 firmware samples and performed a large-scale analysis on 899 firmware images that showed the feasibility of scaling.

Other attacks for IoT devices are also reported, which focus on memory corruption and authentication bypass [35, 38, 66, 89, 103]. For instance, *Firmalice* presented on [89] proposed a model to discover authentication bypass ('backdoors') on embedded devices using advanced program analysis techniques of analyzing binary code. *IoTFuzzer*, given on [35], performs taint analysis in Android applications for IoT devices with fuzzing input attacks to discover memory corruptions that lead to vulnerabilities in the IoT device. Y. David et al. [38] developed a tool called *FirmUp* that performs a static detection for finding common vulnerabilities and exposures (CVEs) in firmware images. Firstly, *FirmUp* lifts the binary to an IR form and extracts the procedures and basic blocks. Then, it uses various techniques to generate and compare the procedures from firmware images to vulnerable procedures from CVEs with great accuracy. From publicly available firmwares, the study found 373 vulnerable procedures, 147 of them to be in the latest available firmware version.

## 3.2. Misuse of Cryptographic Functions

Developers tend to misuse the correct usage of a cryptographic function to achieve the best security possible, even though they are using well-established cryptographic libraries with well-defined APIs. Over the years, studies on misuse of cryptographic functions are reported [27, 56, 83], mainly focused on Android [33, 41, 67, 87, 95] and iOS applications [43, 61].

In 2013, M. Egele et al. developed CryptoLint [41] that performs large-scale static analysis techniques (static program slicing) on Android applications to capture common cryptographic misuses (IND-CPA and cracking resistance). The study found that 88% of the evaluated applications use cryptography inappropriately and violate at least one rule from 6 common cryptographic misuse rules. Those rules also covered in Chapter 2.3 are marked as usage of ECB mode (*Rule S2*), usage of constant IV (*Rule S3*), usage of constant encryption keys (*Rule S1 and K1*), usage of constant salts (*Rule K2*), usage of fewer than 1000 iterations for PBE (*Rule K3*) and usage of static seed for PRNG (*Rule R1*). NativeSpeaker [95] also performs a large-scale static taint analysis on Android native code libraries for cryptographic misuse detection and suggests that third-party libraries are responsible for the misuses. A newer study from I. Muslukhov et al. [67] developed a tool called BinSight that performed a similar study on Android applications between 2012 and 2016. The study showed that the usage of ECB mode has been significantly reduced over the aforementioned years. However, the use of static IVs and keys increased while having an improvement on PBE (unique salt and more iterations) as well as not providing a static seed on PRNG. J. Feichtner et al. [43] performed a similar case study on cryptographic misuse in iOS applications in 2018 and found that 82% of their evaluation dataset (417 apps in total) have at least one violated rule.

A different study from J. Li et al. designed K-Hunt that can discover insecure cryptographic keys by analyzing how they are generated, propagated and used on symmetric, asymmetric, stream ciphers and digital signatures. They evaluate only a single platform, x86/64 stripped executable binaries, on real-world examples using well-established cryptographic libraries such as Nettle, WolfSSL, etc. K-Hunt implements a function-level variant of dynamic taint analysis using Intel PIN [62], a dynamic binary instrumentation (DBI) framework and discovered 22 insecure keys out of 25 evaluated programs. In addition, research on RSA padding identification methods in IoT firmware images was performed by Chao Mu et al. [65] in a dataset of 159 successfully unpack firmware images from 6 different vendors. They performed static code analysis (lifting in an IR form) on executable binaries that dynamically linked the OpenSSL library, aiming to identify which RSA padding schemes are used. IDA Pro [49] is used for function identification, and Angr Framework [1] for translating specific code blocks into IR expression constant analysis to track RSA padding function arguments. Their conclusion is that flawed RSA padding is still in use for IoT environments.

The most relevant work to our study is performed by L. Zhang et al. [102], who designed and implemented a framework called CryptoRex. To the best of our knowledge, they performed the first automated and large-scale analysis to identify cryptographic misuses with a focus on IoT device firmware images. Similar to our work, they first implemented a crawler that captured a total of 1327 firmware images from 12 different IoT vendors and successfully unpacked 521 of them (39.3%). Then, static code analysis is executed only on binaries that are using (dynamically linked) a cryptographic library of their interest, covering 7 well-known including OpenSSL [14] (libcrypt), GnuPg [6] (libgcrypt) WolfSSL [17] (wolfcrypt) and many others. Due to multiple architectures and the diversity of firmware images in IoT devices, the binaries are first lifted to VEX IR and its Python bindings. To disassemble the binaries, Angr Framework [1] is used, while to enhance the conversion, an IDA Pro [49] python recover script is implemented. Afterwards, on each binary, they constructed the inter-procedural control flow graph on each entry point and then constructed the cross-file call graph in order to capture self-defined library wrappers on crypto APIs. CryptoRex is able to dynamically update the list of crypto APIs if a self-defined crypto API wrapper is discovered. For the final step, they performed a static taint analysis with backward tracking on relevant API cryptographic calls to track their inputs. In total, CryptoRex can track a total of 190 crypto-related arguments from 165 crypto APIs. At the end of taint analysis, CryptoRex checks the track inputs (tainted sinks) for any violation of the 6 common cryptographic misuses as covered by M. Egele et al. [41]. Their evaluation shows 24.2% (126/521) of the total unpack firmware images violated at least one misuse rule. In particular, ECB mode is violated 20.5%, constant IV 4.6%, constant keys 11.3%, constant salts 10.8% and no violation on static seed for PRNG.

# 4

# System Architecture and Implementation

This chapter contains a comprehensive explanation of our implemented tool that is used for analyzing the security of firmware images. Figure 4.1 depicts a high overview of the system architecture containing all individual modules (each module is numbered with abbreviation *'M'*), with each one being responsible for a specific task. Each module is explained further in dedicated sections throughout the rest of this chapter. Our work uses only open-source software, as it is possible to implement a complex firmware security analysis tool without using any proprietary software (closed-source). Examples of the open source programs that were used are Binwalk [54], *Ghidra* [70], CVE binary tool [51] and many others. Additionally, we implement many modules using mainly Python scripts and various libraries. The developed tool called *E*mbedded *B*inary *A*nalysis *T*ool (`EBAT`) and is provided open source[1].

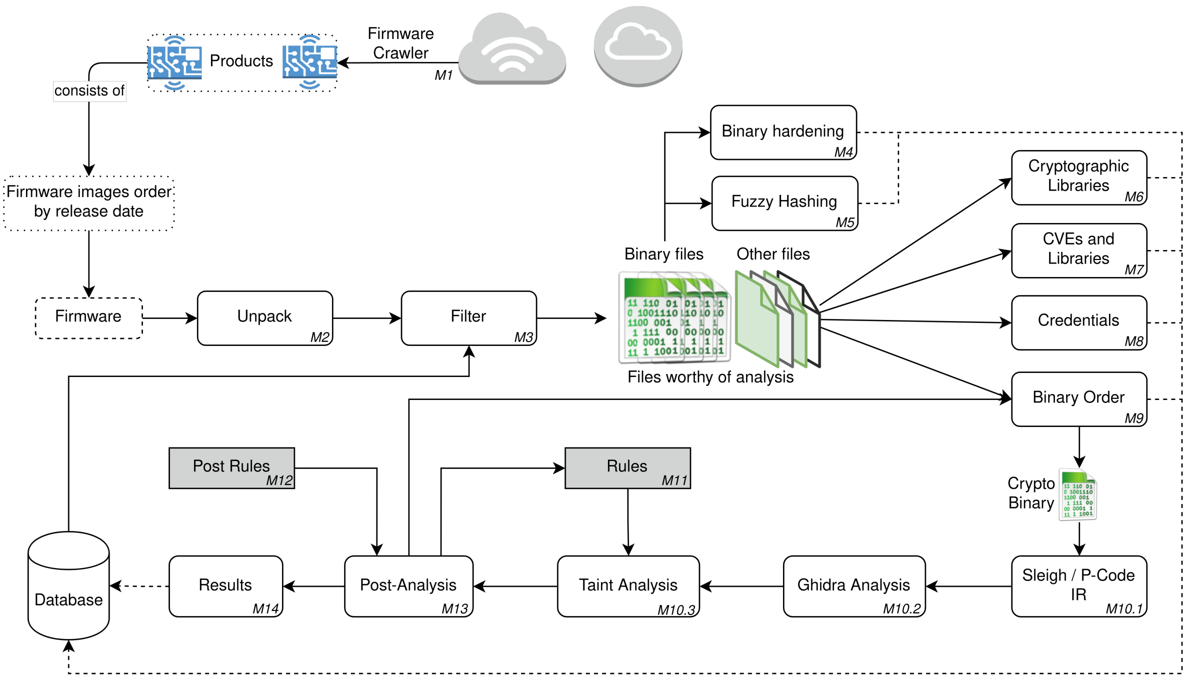



Figure 4.1: Pipeline process of `EBAT's` whole system architecture

To thoroughly assess the security of numerous firmware images on a large scale, it is essential to establish an automated mechanism. This procedure will enable us to obtain a valid, diverse and extensive dataset of publicly available embedded/Internet of Things (IoT) firmware images. Module *M1* is responsible for solving this problem using a crawler that searches for various types of embedded/IoT firmware images from numerous vendors (Appendix table A.4 provides the complete list of crawled

[1]`EBAT` is provided open source at EBAT-public, `https://github.com/ppanagiotou/EBAT-public`

vendors). Along with the crawler, a considerable amount of manual effort is spent to remove potential outliers in the final firmware image dataset, for instance, software for configuring the device and other non-firmware files. Overall, the dataset is organized into multiple products, each containing one or more publicly available firmware images arranged in chronological order from the initial release date to the latest one available at the time of crawling.

The input to `EBAT` is an IoT/embedded product that contains one or more firmware images ordered by their release date. The output produced by `EBAT` is stored in an SQL database, along with individual files that are stored in multiple directories based on the user's arguments for further analysis. `EBAT` starts execution from the firmware image with the earliest release date and automatically continues executing subsequent firmware images one by one until it reaches the latest one. Firmware images are often packaged as compressed archives (and in many other formats) that must be unpacked before analysis. When unpacked, firmware images may produce multiple files that contain and are not limited to executable code, libraries and other resources. In `EBAT`, these files are recognized and stored in different directories to facilitate further analysis based on their file types and contents. Furthermore, duplicated files are not analyzed twice in `EBAT`, which helps reduce the computational power required for analysis. This is particularly relevant for IoT/embedded products that may have multiple firmware images released over time, with each firmware image being an update of the previous one. In such cases, the firmware images tend to have many identical files, which the tool takes advantage of by performing a single analysis for these files. The tool also attempts to optimize memory and computational resources through thread-level parallelism during the analysis of each module.

The analysis of each product's firmware image begins with the unpacking module, *M2*. This module performs a critical step in the whole pipeline, as it unpacks the firmware image to extract individual files that will be analyzed in subsequent modules. The analysis process stops if the unpacking process fails due to firmware obfuscation techniques or encrypted firmware images. Therefore, if the unpacking process fails to extract any files from the firmware image, the user will be notified, and the analysis for that particular image will be terminated. Subsequent to the unpacking module is the filter module *M3* that is responsible for filtering and organizing the extracted/unpacked files to groups of binaries[2] and other cryptographic-related files such as credentials, password files, configuration files, etc. The Filtering module also updates the list of files that have been analyzed from the database and ensures that only new files will be analyzed further.

The modules *M4*, *M5*, *M6*, and *M7* are executed concurrently for each binary file to speed up the analysis process. Module *M8* is also executed in parallel for both binary and other types of files. A brief explanation for the aforementioned modules follows:

- The Binary hardening features module *M4* detects the presence of various hardening features in the firmware image, such as Address Space Layout Randomization (ASLR) and Stack Canary protection. It also checks for other security features like read-only data and code sections. The module saves the results on the database, which can help identify potential security weaknesses in the firmware later on.

- The Fuzzy hashing module *M5* performs fuzzy hashing on each binary file to identify any changes or similarities between subsequent firmware images, which can help identify potential areas of concern.

- The Cryptographic Libraries module *M6* is responsible for discovering the actual version of a cryptographic library embedded in a firmware image. The discovered version of a cryptographic library embedded in a firmware image enables researchers to identify any known vulnerabilities or exploits associated with that version.

- The CVEs (Common Vulnerabilities and Exposures) and Libraries module *M7* is responsible for finding CVEs from all discovered libraries, including the cryptographic libraries, such as *libgcrypt, libssl, libjpeg-turbo, etc.*, that may be embedded in a firmware image.

- The Credentials modules *M8* is responsible for finding several credentials that may be in plaintext and/or embedded on any type of file, including binaries.

[2] `EBAT` focuses only on Executable and Linkable Format (ELF) binaries. Portable Executable (PE) and raw format binaries are left for future work.

The Binary Order module *M9* starts its execution after all the aforementioned modules have finished their tasks. Module *M9* is responsible for ordering and filtering the binaries worthy of static taint analysis to discover any cryptographic misuse. The binaries of interest are the ones that have dynamically linked open-source cryptographic libraries, as will be presented in later sections. Binaries with statically linked libraries are not handled and are left for future work. Additionally, the binaries are grouped into libraries and executables, where libraries are analyzed first in a specific order to discover any potential library wrappers. More details are given throughout this chapter.

For each binary worthy of analysis, various modules are executed. Module *M10.1* is responsible for lifting the binary to *Ghidra's* intermediate representation (*IR*) language, followed by *Ghidra* analysis module *M10.2* that performs various analysis techniques, such as disassembly, function identification, stack analysis, etc., as explained briefly in Chapter 2.2.2. Afterwards, the Static Taint Analysis module *M10.3* is executed, where it performs backward tracking on the function's arguments given from the Rules module *M11*. Rules module *M11* holds all the functions and function arguments of our interest harvested from the open-source cryptographic libraries in order to identify cryptographic misuse rules that are covered in Chapter 2.3. The output of the Static Taint Analysis module *M10.3* is parsed by the Post-Analysis module *M13* that is responsible for translating Abstract Syntax Trees (ASTs) to valuable results that may or may not cause a cryptographic misuse. Post-Analysis Module *M13* uses the Post-Rules module *M12* that holds additional meta-rules, such as mapped values from various cryptographic functions, in order to enhance the translation process of the cryptographic misuse rules. In addition, the module is responsible for appropriately updating the Rules module *M11* in the case of finding a cryptographic library wrapper. Subsequently, the Results module *M14* evaluates the cryptographic misuse condition, if it is violated or not, and saves all the relevant results to the database. Furthermore, this module tries to identify cryptographic primitives, algorithms and parameters used for each firmware's binary. Lastly, all the aforementioned modules' results, as depicted in Figure 4.1, are saved to a database to analyze them later on and produce useful findings, as will be presented in a later chapter.

## 4.1. Firmware Crawler module *M1*

The Firmware Crawler module aims to obtain a valid, diverse and extensive dataset of publicly available embedded/Internet of Things (IoT) firmware images along with their corresponding release dates. A crawler is implemented for every popular vendor site using the open source *Scrapy* framework [86] and various plugin extensions such as *scrapy-splash* for JavaScript support. In addition, firmware images are harvested from publicly available file transfer servers using the File Transfer Protocol (FTP). The crawler is designed to search specific vendor sites of interest to retrieve firmware images and their corresponding release dates. This approach ensures that the crawler only collects relevant information, as globally searching the entire internet for firmware images would be inefficient and time-consuming. By targeting specific vendor sites, the crawler can efficiently gather the necessary data for analysis.

Additionally, this approach allows for better control and monitoring of the data collection process, as the crawler can be adapted to each vendor's specific site structure and requirements. Appendix Table A.4 presents the complete list of vendors. To further reduce the amount of crawled data, a filter is implemented to discard non-firmware files such as documentation, user manuals, other software files, etc. The implemented crawler strictly follows the rules set out in the '`robots.txt`' file of each vendor's site and only downloads publicly available firmware images and their corresponding release dates. In cases where a vendor has a few firmware images, we download the images manually.

Our crawlers may produce false positives, including files incorrectly marked as firmware images and inaccurate firmware image release dates. To ensure the accuracy of our dataset, we manually reviewed the output and removed any non-firmware image files while also correcting any inaccurate firmware release dates that we encountered during the crawling process. We grouped the downloaded firmware images for each vendor into products, each containing firmware images and their corresponding release dates. While we attempted to group the products into different device classes, it was challenging due to each vendor's varying naming approaches and schemes. Additionally, finding the release date for each firmware image using a crawler was a tedious task and was not always possible. Hence, we discovered the release date following the steps below:

1. Crawl the release date from the vendor's website.
2. If it is not found, discover it manually from the vendor's website.

3. If it is not found, extract it from the firmware's image metadata.

We perform additional steps to reduce the overall dataset size and limit the need for computational and storage resources. We remove duplicates for each product and combine identical firmware images of different products. We also configure our crawlers to discard any files that have already been downloaded and filter out any possible non-firmware images. Before removing duplicates, we keep all the relevant information of identical files in the dataset database. The dataset comprises *5-tuple* elements, with each element queued for analysis.

```
(Vendor, Product Name, Product Type, {Firmware images}, {Release dates})
```

## 4.2. Unpack Firmware module *M2*

As described in Chapter 2.1, in order to analyze a firmware image, the first step is to unpack it successfully. Due to a lack of standards, each vendor may use its own packed/unpacked procedures. In addition, vendors may use firmware obfuscating techniques, monolithic firmware images, or even encrypted images that prevent any unpacking process without decrypting them first. Many tools have been developed to overcome some of the issues above, such as *Binwalk* [54], *BANG* [2] and *FACT extractor* [46]. `EBAT` uses mainly *Binwalk* [54], a state-of-the-art unpacking tool for firmware images. In our initial experiments, it has the highest successful unpacking rate from a small initial sample. Along with *Binwalk*, a recursive approach is implemented with additional Python modules for extracting specific file types (e.g., squashfs, cramfs). The recursive approach has the benefit of unpacking as many files as possible; however, it comes with the disadvantage of being time-consuming and sometimes imprecise. To further reduce the number of incorrect extracted files, a filter is implemented in every unpacking stage that marks the already extracted files and removes any potential duplicates. At the same time, it also filters the already known file types using mime types (magic bytes).

*Overall, the approach used by `EBAT` is as follows:* Firstly, the unpacking process starts with searching and decompressing many of the publicly known compressed file formats such as 7z, zip, tar, etc. Secondly, *Binwalk* is used in recursive *("matryoshka")* mode for extracting each previously produced file along with the implemented Python modules. All the steps are recursively run until no new files can be produced. Additionally, the filter mentioned above is used in all stages. Note that *Binwalk* uses signature file carving techniques (through magic headers) that occasionally may lead to very large unreliable and incorrect file outputs. In order to prevent that, a fail-safe is implemented that stops the Binwalk process and continues the analysis with the files that are successfully unpacked. Lastly, to decide if a firmware image is successfully unpacked, a heuristic is implemented that considers the validity of the extracted binaries. Specifically, during the unpacking process, we determined the success of the unpacking based on the presence or absence of binaries. If no binaries are found, the firmware is marked as not successfully unpacked. Conversely, if at least one binary is found, a check is performed on the dynamic libraries needed by the binary. The firmware is marked as successfully unpacked if all of these libraries are found. However, if any of these libraries are not found, the firmware is marked as partially unpacked.

## 4.3. Filtering module *M3*

The filtering module is responsible for walking through the unpacked files, identifying whether they are worthy of analysis and passing them to the relevant modules. This module has the advantage of limiting the unnecessary processing power while it passes files to modules that will likely provide us with useful findings/results. `EBAT` focuses on two types of files: ELF binaries and cryptographic-related files. For filtering and discovering those files, various open-source tools and Python modules are used, some for discovery and others for verification. Examples of such tools are *readelf* [3]*, ssh-keygen* [4]*, openssl [14], yara [94]*.

Cryptographic-related files are any kind of file that may consist of or related to one or more cryptographic operation(s), such as:

- Credentials, for example, *certificates, public keys, private keys, ssh keys,* etc.

[3]"readelf(1) - Linux man page", Linux, `https://linux.die.net/man/1/readelf`
[4]"ssh-keygen(1) - Linux man page", Linux, `https://linux.die.net/man/1/ssh-keygen`

- Hash passwords, for instance in *passwd, shadow (Linux file user/password file) configuration files,* etc.
- Configuration files, such as, *ssh, web configurations,* etc.
- Script files that may consist of a cryptographic operation, such as `openssl enc -aes-256-cbc -salt`.

For each ELF binary, `EBAT` discovers the followings:

- File Type: *ELF, PE, RAW* format.
- Type: Executable or Library.
- Architecture: *ARM 32 bit, MIPS 64 bit, x86,* etc.
- Endianness: Little endian (*LE*) or Big endian (*BE*)
- Set of Dynamically Linked Libraries.

Additionally, an ELF binary can have from zero to many dynamically linked libraries (shared libraries). The filtering module discovers and separates the binaries that use no cryptographic libraries from those that use at least one cryptographic library. Those are marked as binaries worthy of static code analysis and will be used later in other modules. The overall cryptographic libraries the tool handles are depicted in Section 4.6, Table 4.1. The binaries that are deemed worthy of analysis, also called '*crypto binaries*', are analyzed further with *Ghidra SRE* [70] using static taint analysis on multiple functions and function arguments. Lastly, the set of crypto binaries may change as the analysis progresses due to library function wrappers that may be discovered. More details on this are given in the Binary Order module *M9* (Chapter 4.9).

## 4.4. Binary hardening module *M4* - Exploit Mitigation Indications

Many exploitation techniques, such as buffer overflow attacks, integer overflow attacks, and stack smashing attacks, are made possible by the absence of binary hardening features in firmware images as reported in Chapter 2, Section 2.2. Using binary hardening features is an effective way to improve firmware security by making exploitation harder for adversaries. It is an essential step in enhancing the overall security of embedded devices. This module aims to determine whether or not exploit mitigation indications are present in the set of binaries within a firmware image. `EBAT` analyzes each binary separately for hardening features using the *hardening-check*[5] and *readelf* linux tool[3]. Furthermore, the exploit mitigation indications are saved to the database. It should be noted that this module finds an *indication* of the exploit mitigation techniques and may raise false alarms in some indications. Each of the hardening techniques is described briefly below:

- **Position Independent Executable (PIE)**: PIE is an indication that the 'text' section (program's code) of the binary can be relocated somewhere in memory. Address Space Layout Randomization (ASLR) security technique must be supported by the executing kernel in order to take full advantage of PIE. In rare cases, PIE may be enabled, but the detection algorithm could fail to recognize it due to specific characteristics in the binary structure or the firmware image.
- **Non-Executable Bit (NX)**: The NX bit indication marks memory regions as non-executable. This technique prevents an adversary from executing code in arbitrary memory regions.
- **Stack protected (Stack Canaries)**: Stack protected mitigation provides resistance against stack buffer overflow attacks. *Stack Canaries* are special bytes of sequences in memory that are checked for changes during run-time. When a function is called, a canary value is placed on the stack before the return address. The canary value is then checked before the function returns to ensure it has not been modified. If the canary value has been modified, it means that a stack buffer overflow attack has occurred, and the program will terminate. The identification

[5]"Ubuntu Manpage: hardening-check - check binaries for security hardening features", Canonical Ltd. Ubuntu, https://manpages.ubuntu.com/manpages/focal/man1/hardening-check.1.html

of the stack canaries from this module indicates that the binary is compiled with stack protector enabled. It may provide false alarms if no array is being allocated on the stack and the ELF binary is compiled with stack protected options.

- **Fortify Source functions**: When compiling a binary, the compiler will try to replace unsafe libc functions with their safer counterparts using Fortify Source binary hardening, e.g., `strncpy` instead of `strcpy`. This mitigation technique prevents buffer overflow attacks due to the usage of the safer counterpart functions, most of which require additional arguments such as length. There is a possibility of false alarms as the check will pass only if any fortified function is found and will fail only if unfortified functions are found.

- **Read-only relocations (RELRO)**: RELRO marks any regions in the Global Offset Table (GOT) as *read-only* that are already resolved before the execution begins. Thus, it reduces the memory region of a binary that an adversary can use to perform a successful memory corruption exploit. This technique is also called *partial RELRO*. When combined with Immediate binding (see below), it additionally reduces the ability of an adversary to execute a successful *GOT overwrite* attack (also called *full RELRO*).

- **Immediate binding**: Immediate binding indicates that the run-time linker must perform all relocations before the program executes (the opposite is called *Lazy binding*). Thus, all memory locations from shared libraries or global variables are marked as *read-only* compared to *partial RELRO*, which marks only the already resolved relocations. Combined with *partial RELRO*, as described above, this is also called *full RELRO*, which further reduces the memory region an adversary can use to perform a memory corruption attack.

## 4.5. Fuzzy hashing module *M5*

This module's objective is to calculate a SHA-256 digest [77] and a fuzzy hash signature called *ssdeep* [53] on every discovered binary. By computing the SHA-256 digest of each binary file, we can determine if a binary file from one firmware image is identical to a binary file from another firmware image. The digests are useful for identifying if a binary has any modifications between firmware versions. On the other hand, the fuzzy hash signature calculates a similarity score between two binaries based on their content. By calculating the *ssdeep* hash signature for each binary in a firmware image, we can compare it with the *ssdeep* hash signatures of other firmware images to identify potential similarities or changes between them.

The one-way hash function (SHA-256) shows if a binary is entirely identical, byte by byte, to any other binary. On the other hand, *Ssdeep* is a program for computing context-triggered piece-wise hashes (CTPH) [53], also called fuzzy hashes, which can match binaries that have sequences of identical bytes in various orders that might differ in both content and length. It returns a hash signature for each binary that can be compared with other *ssdeep* hash signatures and will provide us with a score value from 0 to 100. The hash signature indicates a matching score between two hash signatures where a zero indication means that the binaries did not match at all, and a 100 indication means that the binaries are an identical match. For instance, consider 2 binaries `A` and `B`, where binary `A` differs one line of code from binary `B`. The score value calculated from *ssdeep* compare function for these binaries will be near to 100 but not equal since it is not an identical match (the digest from SHA-256 will be completely different). Finally, the two digests *SHA-256* digest and *ssdeep* hash signature are saved into the database.

## 4.6. Cryptographic libraries module *M6*

The cryptographic libraries module *M6* is responsible for discovering cryptographic libraries and their version from a firmware image. `EBAT` analysis focuses on 12 cryptographic open source C/C++ libraries that are widely used on firmware images, many of them presented in [102]. There are cases in which the actual library binary may not exist on our unpacked set of binaries, probably due to partial unpacking. However, it is linked dynamically to an executable binary. The cryptographic libraries are discovered on binaries only if they are dynamically linkable, whereas static libraries are left for future work. Furthermore, this module has the ability to discover the actual version of 8 out of 12 cryptographic libraries as marked with symbol '✓' in Table 4.1. Along with the actual version, it may also discover the library's CVEs using the appropriate Common Platform Enumeration (CPE) structure, extracted from the National Vulnerability Database (NVD) and provided by NIST [74].

In order to find the accurate version of a cryptographic library, a *Ghidra* headless script [70] is implemented. This script taints the appropriate export function(s) that is responsible for returning the library version. The returned value is marked as a taint source, and the script performs backward tracking in order to find the constant version (tainted sink). At last, it returns the discovered version to the main tool. For instance, in the WolfSSL library, the script taints the returned value of `'wolfSSL_lib_version'` function (taint source), where it tries to discover and return the version (taint sink). In this particular function is a string element, e.g., `"4.4.0"`. Another example is in the OpenSSL library where the tainted functions are '`OpenSSL_version`' and/or '`SSLeay_version`' depending on the OpenSSL version. Finally, a list of tainted functions is created for the `8` cryptographic libraries so that a version can be successfully recovered.

Other tools, such as the CVE bin tool [51] and FACT [45], may provide a library version based on heuristic methods on strings and yara signatures, respectively. In contrast, our technique (`EBAT`) uses code analysis to discover a library version only for cryptographic libraries. While our method is more computationally expensive, it may provide better accuracy in terms of detecting the actual version. However, there are cases where our script may fail to find the actual version, particularly when there is no appropriate version function compiled with the library (stripped).

Table 4.1 depicts the various cryptographic libraries that this module is able to discover. The symbol '✓' represents the cryptographic libraries in which the *Ghidra* headless script attempts to detect the actual library version. With the symbol '✗', no version recovery is implemented, mainly due to the non-existence of the return version function call (found in our experiments and left for future work). The chosen libraries that this module discovered are the most commonly used in our experiments except for *Libc (uClibc-ng [16] or glibc [5]) [48] 'libcrypt'*, where, a version return function does not exist.

| | Name | Library | Discovered version |
|---|---|---|---|
| 1 | Crypto++ [3] | libcrypto++, libcryptopp | ✗ |
| 2 | GnuPG [6] | libgcrypt | ✓ |
| 3 | GnuTLS [7] | libgnutls | ✓ |
| 4 | KerberosV5 [93] | libk5crypto | ✗ |
| 5 | Libc (uClibc-ng [16] or glibc [5]) [48] | libcrypt | ✗ |
| 6 | Libsodium [9] | libsodium | ✓ |
| 7 | LibTomCrypt [10] | libtomcrypt | ✗ |
| 8 | mbedTLS/PolarSSL [11] | libmbedcrypto, libmbedtls libpolarssl, libmbedx509 | ✓ |
| 9 | Mcrypt [12] | libmcrypt | ✓ |
| 10 | Nettle [13] | libnettle | ✓ |
| 11 | OpenSSL [14] | libcrypto, libssl | ✓ |
| 12 | WolfSSL [17] | libwolfssl, libcyassl | ✓ |

Table 4.1: Cryptographic Libraries discovered by module *M6*

## 4.7. Common Vulnerabilities and Exposures (CVEs) and Libraries module *M7*

The CVEs and Libraries module *M7* is capable of finding common vulnerabilities and exposures on binaries (executables and libraries and cryptographic libraries that are discovered by the previous module *M6*) inside a firmware image. `EBAT` mainly uses the CVE Binary Tool provided by Intel [51]; "This tool scans for a number of common, vulnerable components such as openssl, libpng, libxml2, expat and a few others, to let you know if your system includes common libraries with known vulnerabilities". It uses the strings discovered on binaries in order to extract library signatures and version numbers with heuristic methods. However, it may provide false positives (incorrect versions) if the signature match failed or if it was intentionally obfuscated and also false negatives where it is unable to discover the actual library version. Furthermore, it uses the National Vulnerability Database (NVD) provided by NIST [74] to cross-reference the discovered version with any known CVE using the appropriate Common Platform Enumeration (*CPE*) structure [73].

For every extracted binary in a firmware image, the CVE Binary Tool is executed. The execution may provide the library or binary version, excluding the cryptographic libraries that are discovered by the previous module *M6* and possible CVEs. Furthermore, Yara signatures provided by FACT [45] are also used for finding additional library and executable versions such as the *kernel version, busybox,* etc. The *CPE* (Common Platform Enumeration) is created through the following steps: querying the NIST database for CVEs by utilizing the identified version and extracting the corresponding CVEs using the csv2cve tool [51]. Subsequently, these extracted CVEs are stored in our database for further analysis and reference.

## 4.8. Credentials module *M8*

Credentials play a significant role in embedded/IoT firmware image security. Having a private key in plaintext is a major security issue and should be avoided. This module can discover two types of credentials. Firstly, those that are in a file format such as *PEM* (Privacy Enhance Mail), *CRT* (certificates), *CSR* (certificate signing request), private and public keys, *SSH* keys, etc. Secondly, embedded credentials in binaries and/or other files are discovered using Yara signatures provided by FACT [45].

Table 4.2 depicts the types of credentials that this module can identify. Before being saved to the database, the validity of these credentials is verified using various cryptographic tools, including *openssl* [14], *ssh-keygen*, *pgpdump* [99], and *gpg* [6]. Rather than relying on magic types, these tools perform file structure analysis to ensure the validity of the discovered credentials. Additionally, some credentials may be encrypted using a password or key. The module tries to decrypt them using known passwords as well as passwords that were manually discovered by analyzing various firmware image files. The complete list of passwords can be found in Chapter 5.4.

| Various Types | Common files extensions[6] |
|---|---|
| Certificates | *.cert, .crt* |
| Private Keys | *.key, .pem* |
| Public Keys | *.pub, .pem* |
| Various cryptographic Parameters | *.pem,* |
| Certificate Signing Requests | *.csr* |
| SSH Public Keys | *ssh_rsa_host_key, dropbear_rsa_host_key* |
| SSH Private Keys | *ssh_rsa_host_key, dropbear_rsa_host_key* |
| PKCS12 file formats | *.p12* |
| PGP, GPG | *.gpg* |

Table 4.2: Types of Credentials discovered by module *M8*

## 4.9. Binary Order module *M9*

The Binary order module *M9* aims to filter and find the order of the set of binaries extracted from a firmware image that will be used for static taint analysis, which aims to discover cryptographic misuses. The module acts as a filter for all binaries and passes only those that use at least one dynamically linked cryptographic library provided by Table 4.1 or a cryptographic library wrapper, as will be explained later. Those binaries are called in this context 'crypto binaries' and are worthy of code analysis by `EBAT`. Crypto binaries are divided into two categories, executable and library binary, that use one or more dynamically linked cryptographic libraries.

The order of analysis of a binary is essential only for library binaries since the functions of the analyzed library may later be used in an executable binary. Those functions are also referred to as *wrapper functions*. Thus, the first step toward binary analysis is to analyze only the library binaries in order to determine any API cryptographic misuse wrapper functions that may be called later on by an executable binary. To determine the specific order of analysis, a directed graph $G = (V, E)$ is created with the following:

1. A vertex $V_x$ can represent either a library binary or a cryptographic library.

2. A directed edge $e_1 = (V_1, V_2)$ indicates that node $V_1$ has a dynamically linked library node $V_2$.

3. The set of vertices $V$ includes the discovered library binaries that use a cryptographic library and the actual cryptographic library itself.

4. The set of edges $E$ is created from the dynamically linked libraries of each binary as directed edges. For example, $E = \{(V_1, V_{d_1}), (V_1, V_{d_2})\}$, where $V_1$ has dynamically linked libraries $V_{d_1}$ and $V_{d_2}$.

Once the directed graph is created, we can use the topological sort algorithm to determine the order in which the library binaries should be analyzed. However, a circular dependency can occur when two or more libraries call each other. In this case, we remove the nodes that cause the circular dependency and recalculate the topological sort order. These nodes (library binaries) will be analyzed first since there is no other way to proceed. Algorithm 1 provides a brief overview of the steps in determining the analysis order. The input to the algorithm is the set of all binaries, including libraries, executable and cryptographic library binaries, that have not been analyzed yet, and the output is the binaries that will be analyzed in a specific order. It is worth noting that a library may create function wrappers that may be used in an executable binary, meaning new binaries may also be worthy of code analysis. In such cases, we run Algorithm 1 recursively until no new binaries that are worthy of code analysis are generated.

**Algorithm 1:** Producing the order of the analysis

**Result:** Order of the analysis
**Input:** $L \longleftarrow$ set of libraries
$C \longleftarrow$ set of cryptographic libraries
$E \longleftarrow$ set of executable binaries
$L \longleftarrow L \cap C$;
$E \longleftarrow E \cap C$;
$OL \longleftarrow ProduceOrderofLibraries(L)$;
return $OL, E$;
**Function** `ProduceOrderofLibraries` $(L)$:
  $G \longleftarrow GreateGraph(L)$ ; `// Create the libraries graph`
  **return** $ProduceTopoSort(G)$ ; `// Compute the topological sort`
**End Function**

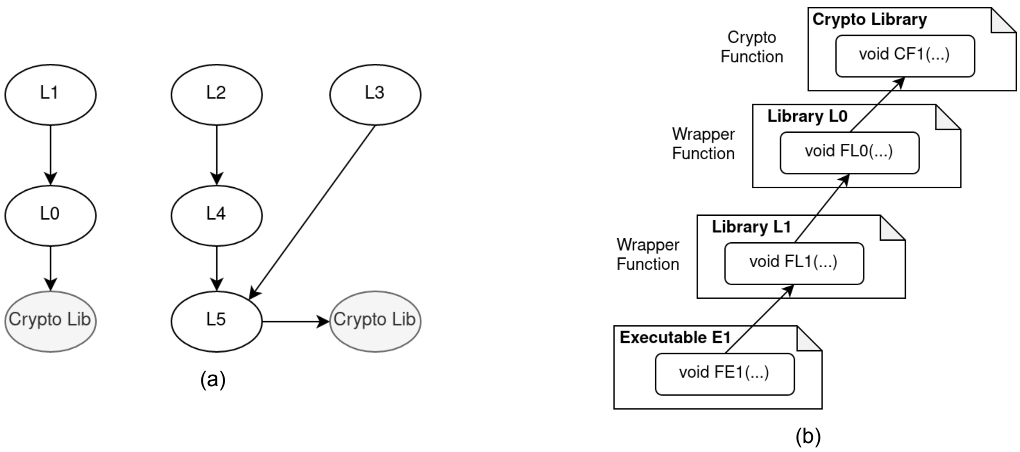



Figure 4.2: Figure (a) shows an example graph produced by dynamically linked libraries and Figure (b) presents an example of wrapper functions used by libraries reaching the cryptographic function of our interest.

An example of the library graph is illustrated in Figure 4.2a, where the set of nodes are libraries that have dynamically linked a cryptographic library. Library `L0` uses `Crypto Lib`, `L1` uses `L0` and so on. Those dependencies form the set of edges. The topological sort of the graph and hence the order of analysis is the following: Firstly, library `L5` is analyzed, then `L0`, `L3`, `L4` in any order (parallel execution is possible), then `L2` and lastly `L1`.

Figure 4.2b illustrates an example of the discovery and use of function wrappers. Executable binary `E1` uses library `L1` where `L1` is using `L0` and finally `L0` is using a cryptographic library. Hence, a cryptographic function that is called `CF1`. `EBAT` is able to discover and create dynamic rules of every function wrapper, as shown in the example. `FE1` is using `FL1` function (wrapper function), where `FL1` is using `FL0`. Finally, `FL0` uses the function of our interest that must be tainted.

In order to taint all prior function wrappers, `EBAT` needs to analyze the `L0` library in order to discover the appropriate wrapper function `FL0` that uses the cryptographic function of our interest `CF1`. Then, create the function rule and proceed to the analysis of `L1` to find the `FL1` function wrapper and finally to the `E1` executable binary to find the `FE1` function that provides the values of arguments in order to taint it and perform our taint analysis successfully. With that order, the cross-file called graph is built, and all the function wrappers that may contain a misuse rule and/or identifications of cryptographic primitives are updated. Furthermore, the filter is recursive. Consequently, `L0` is firstly analyzed as it uses a cryptographic library. Then, all libraries that use `L0` must be added to the set of binaries worthy of analysis, so `L1` is added. Additionally, all the executable binaries that are using `L0` and `L1` must be added as well. With this methodology, `EBAT` can analyze all possible binaries that may use a cryptographic function wrapper or not.

## 4.10. IR module *M10.1* and *Ghidra* Analysis module *M10.2*

The *Ghidra* Analysis module *M10.2* utilizes *Ghidra SRE* [70], an open-source software reverse engineering (SRE) suite of tools provided by the National Security Agency (NSA). `EBAT` leverages the *Ghidra* headless mode, which allows for the execution of headless scripts without user interaction. The first step in the analysis process is to import the binary into *Ghidra* and use its out-of-the-box analysis. The first step of this analysis disassembles and converts the binary's assembly language instructions into *Ghidra's* intermediate representation/language (IR/IL) form, known as *P-Code* (IR module *M10.1*). The benefits of using an IR form are explained in Chapter 2.2.2.

Although *Ghidra* provides out-of-the-box analysis for binary files, not all processor architectures are supported by it. However, users can expand the range of supported architectures by creating translations from machine instructions to P-Code using the SLEIGH language. Once the binary is lifted from assembly language instructions to P-Code (IR module *M10.1*), the analysis proceeds through several steps, including function identification, data-flow analysis, function and data reference analysis, stack and address tables creation, demangler, control flow analysis, type analysis, cross-referencing analysis and many others. A prescript can be written to choose from various analysis options to customise the analysis for a particular binary. By default, the *Decompiler Parameter ID* option, which creates parameters and local variables for a function, is disabled due to the significant amount of time it takes to execute. However, this option, along with others that may enhance the analysis results, can be enabled using `EBAT`'s arguments at the cost of more computational resources. Overall, the *Ghidra* Analysis module *M10.2* used by `EBAT` performs the following steps:

1. Import the binary into *Ghidra* and disassemble and lifts to IR (IR module *M10.1*).
2. Analyze binary using *Ghidra's* out-of-the-box analysis techniques.
3. Identify the main function on executable binaries. This step can be challenging for stripped binaries, which have removed their symbol. A headless script was developed to identify and mark the main function using the `'start'/'_entry'` point address to address this issue. The script is particularly useful for binaries that are compiled with a C/C++ compiler and use the C standard libraries glibc [5] or uClibc-ng [16].

## 4.11. Rules module *M11*

The *Rules* module *M11* contains the functions and their arguments that are relevant to our analysis[7]. This module serves as an input for the *Static Taint Analysis* module *M10.3*, which will be described later. All functions and function arguments are created according to the well-defined API of the 10 cryptographic libraries depicted in Table 4.3, and in accordance with the 18 cryptographic rules that check for cryptographic misuse as defined in Chapter 2, Section 2.3. The importance of this module is to provide a variety of cryptographic functions to detect cryptographic misuses and cryptographic primitives. The functions and function arguments were initially created based on *CryptoRex* [102], and later on were heavily expanded using useful information from previously analyzed firmware images, either manually or through automated scripts. Additionally, new functions and arguments are discovered from the cryptographic libraries' API documentation. This module is independent of all other modules; thus, users can add/modify any function and their corresponding arguments of interest and perform the analysis.

[7]The complete list of tainted functions and functions arguments can be found at EBAT-public rules, `https://github.com/ppanagiotou/EBAT-public/blob/master/configurations/rules.conf`

Function wrappers that consist of a cryptographic function call (see Binary Order module *M9* Chapter 4.9) are discovered and automatically added to the Rules module *M11* as the analysis progresses. Post module *M12* described in Section 4.13 is responsible for the translation and creation of a new rule for every newly discovered wrapper function. Table 4.3 presents the total number of functions and function arguments that are tainted for each cryptographic library for the purpose of identifying a violation of one or more cryptographic rules.

| # | Library | # tainted functions | # tainted arguments |
|---|---|---|---|
| 1 | Crypto++ [3] | 2 | 7 |
| 2 | GnuPG [6] | 24 | 67 |
| 3 | GnuTLS [7] | 2 | 2 |
| 4 | KerberosV5 [93] | 16 | 36 |
| 5 | Libc (uClibc-ng [16] or glibc [5]) [48] | 25 | 27 |
| 6 | LibTomCrypt [10] | 52 | 117 |
| 7 | Libsodium [9] | 14 | 25 |
| 8 | mbedTLS [11] | 116 | 397 |
| 9 | OpenSSL [14] | 369 | 629 |
| 10 | WolfSSL [17] | 113 | 339 |
| - | **Overall** | **733** | **1646** |

Table 4.3: Number of cryptographic tainted functions and functions arguments from each cryptographic library.

The rule format that `EBAT` uses is described for the rest of this section. The function arguments can be declared out of 8 types as listed below. Each value is treated differently depending on the declaration type of argument. In addition, priorities may be defined to arrange which argument must be tainted first. Priorities are in the form of `a<b`, meaning that argument `a` has a higher priority than `b`; thus, it needs to be resolved first. For each rule, multiple priorities may be defined where they are important in terms of context, argument length, and other types of arguments, as their results are needed to continue the analysis further.

**Types of tainted arguments:**

- `int`: treated as signed or unsigned integer.
- `bit`: treated as integer value but with the metadata that this value defines a bit length. Later on, the value will be converted to a byte length.
- `byte`: treated as integer value but with the metadata that this value defines a byte length.
- `string`: string ending with a null terminator (`\0`).
- `bytes`: byte array of arbitrary length.
- `output`: output value mainly a pointer.
- `CTX`: context object.
- `CTYPE`: cipher type treated as int or context object.

For instance, Advanced Encryption Standard (AES) [75] has three key lengths 128, 192 and 256 bits, respectively. The function of our interest does not specify from the function declaration what AES variation is using. Instead, an argument is provided for choosing the key length; for instance, the function declaration of the AES decryption function for setting a key from mbedTLS [11] library is as follows:

```
int mbedtls_aes_setkey_dec(mbedtls_aes_context *ctx,
const unsigned char *key, unsigned int keybits)
```

, where the argument `keybits` must be 128, 192 or 256 to specify which AES variant will be used. `EBAT` rules module needs to account for that. Therefore, the static taint analysis module needs to identify the key length argument of AES first in order to determine which variation of AES key is being used and, therefore, determine the key length. Second, suppose a key constant value is discovered, for example, a pointer to a data segment (*.rodata* section). In that case, the static taint analysis module will extract the correct number of bytes as the key length was discovered earlier. However, if the key length of a function's signature is known, then the rule is created as a predefined constant. Some functions from our cryptographic libraries documentation may define the length with a default value and also provide an argument that may or may not be used. If that is the case, `EBAT` analysis still taints the length argument, and if it is discovered[8], then, the default value is overwritten, and the new key length is used; otherwise, the default value is taken into account.

Below, two examples are given, introducing some corner cases and our approach to solving them. These examples are real-world examples found in firmware images.

- In the first example, we have a function from GnuPG [6] ('`libgcrypt`' cryptographic library) that is responsible for setting a symmetric key. The prototype of that function is given below:

  ```
  gcry_error_t gcry_cipher_setkey (gcry_cipher_hd_t h,
  const void *k, size_t l);
  ```

  In the function above, all arguments must be tainted as all arguments provide useful information on the code analysis. The first argument of type '`gcry_cipher_hd_t` provides information about the context. In our analysis, this type of argument is marked as '`CTX`'. These types of arguments provide useful insights into the underlying algorithm provided by the context, including other API functions that will be used in this context and more. The second argument '`k`' is the key, which is marked in our analysis with type '`bytes`'. Lastly, the third argument, length '`l`', is marked in our analysis with type '`byte`' to indicate that the key length is in bytes. The order of arguments plays a significant role in this cryptographic function. The context '`h`' is resolved at the beginning, providing us with information about the cipher algorithm and possibly the mode of operation (for symmetric key encryption). Afterwards, the key length '`l`' must be found as it will provide the size in bytes. Lastly, the key needs to be resolved, and if it is found as a constant value (pointer to a data segment), then '`l`' bytes of data will be extracted, and a rule is violated.

- A more complicated example is provided by OpenSSL [14] cryptographic library, a widely used function that performs symmetric key encryption. The function prototype is the following:

  ```
  int EVP_EncryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type,
  const unsigned char *key, const unsigned char *iv);
  ```

  Again, all arguments are tainted in our analysis: the cipher context is marked as '`CTX`', the cipher type is marked as '`CTYPE`', the symmetric key '`key`' and the initialization vector (IV) '`iv`' is marked as '`bytes`' type. Firstly, the '`CTX`' must be resolved and secondly, the cipher type '`CTYPE`'. In the OpenSSL library, the '`EVP_CIPHER`' and '`EVP_CIPHER_CTX`' types are objects that are initialized with a function call. Thus, they do not have a primitive value such as an integer but mainly return a function address. The specificity of those arguments needs to be resolved differently than others, whereas it will not lead to a constant value but instead to a function call. Hence, a special case applies to these types of arguments. For instance, if the value of '`CTYPE`' is equal to a function call, e.g., '`EVP_aes_256_cbc()`', with the help of Post Rules module *M12* (see section 4.13) it will provide us all the necessary information to resolve the type, the key length, the IV length and the mode of operation of the underlying cipher. In the example provided, '`EVP_aes_256_cbc()`', the cipher algorithm is AES [75], with block size length of $128$ bits, which results in the IV size length of $128$ bits. Furthermore, the key size is $256$ bits, and the mode of operation is Cipher Block Chaining (CBC) [76]. Thus, the key and IV length can be resolved directly when the underlying algorithm is discovered. If the key and IV arguments are found to be constants, bytes that are equal to the resolved length can be extracted, and rules are violated. Many functions have default values and different behaviours depending on the arguments input. For this specific function, the OpenSSL documentation states that "it is possible to set all parameters to '`NULL`' except type in an initial call and supply the remaining parameters in subsequent calls, all of which have type set to `NULL`. This is done when the default cipher parameters are not appropriate."[9]. The tainted context '`CTX`' solves that, as it will taint all appropriate function calls and update our tainted context

[8] In most cases the argument value will be '`null`' or 0 if it is not in use.

[9] "OpenSSL manual 1.0.2 - *EVP_EncryptInit*", OpenSSL [14], `https://www.openssl.org/docs/man1.0.2/man3/EVP_EncryptInit.html`

metadata only if the same context is used. Therefore, even if ‘NULL’ is used as the key and the key data is resolved at a later stage in a different function call with the same context, the algorithm specification data will still hold true for this context, and the key length is already resolved. Overall, the Post Rules module *M12* defines those meta-rules as precisely as possible for each and every cryptographic function of our interest.

## 4.12. Static Taint Analysis module *M10.3*

The goal of this module is to perform static taint analysis (backward tracking) on each function call argument that is provided by the Rules module *M11* (section 4.11). Taint analysis creates the Abstract Syntax Tree (AST) of P-codes until it determines a constant value (if it exists). All the inter-process communications from *Ghidra* to EBAT main modules are exchanged by JavaScript Object Notation (JSON). Subsequently, the Post-Analysis module *M13* (section 4.13) is responsible for translating and identifying if the discovered AST leads to a possible cryptographic rule violation. The analysis is not sound (everything is marked as a potential vulnerability) and may provide false positives, as well as potential cryptographic misuse, which may be missed (false negatives).

A *Ghidra* headless script, an idea inspired by INFILTRATE 2019 conference [32] and heavily expanded upon, is developed to perform the static taint analysis. In addition to the analysis, the implemented script finds all call sites of the functions provided by the Rules Module *M11* and discovers their references to create the call graph. The taint analysis is *context sensitive*, meaning the order of function calls is taken into account, and a separate result for each possible path is calculated. From the *Ghidra* Analysis module *M10.2*, the program’s main is discovered (for executable binaries). Furthermore, function wrappers from previously analyzed libraries are also taken into account as they will be updated in the Rules module *M11*. Thus, if the analyzed executable uses a function wrapper, a cross-file call graph (CFCG) is created with the additionally discovered function wrappers. Subsequently, Depth First Search (DFS) on the CFCG is performed to find whether a particular source function is called from the main. The results of CFCG, DFS as well as the AST are saved and processed by later modules. Overall, the script begins with a taint analysis of every discovered function call site and taints each argument individually with respect to the priority order as described in Rules module *M11*.

**EBAT defines the followings:**

- **Taint sources:** Taint sources are marked as the function prototypes; hence, each argument has various types and priorities. The order and type of each argument are treated accordingly. Specifically, the taint source is tagged as the P-code i’th input argument (from the input varnode) of the called function of our interest.

- **Taint propagation:** The script recursively performs backward taint analysis on each taint source. For every P-code operation that is discovered, a node is saved and added to the Abstract Syntax Tree (AST) while creating appropriate edges. This simple taint propagation algorithm builds an AST on each taint source until it reaches a constant taint sink or a P-code operation that cannot be further resolved.

- **Taint sinks:** Taint sinks are defined as pointers or immediate values or as objects that are created from a function call. This is based on the type of tainted argument as declared in Rules module *M11* (section 4.11). Thus, if it is an immediate value (declared as `int, bit, byte` (types of tainted arguments)), then it tries to resolve it instantly, a constant node is created, and the taint propagation stops. Additionally, if it is a pointer (declared as `string, bytes`(types of tainted arguments)) and it is directly resolved to ‘.rodata‘ segment, then a constant node is created, including $x$ amount of bytes of extracted data and the taint propagation ends. For the remaining types (declared as `CTX, CTYPE, output` (types of tainted arguments)), a taint sink is resolved as a pointer reference to a function call that may create the object or all the aforementioned types of tainted arguments. There are cases in the taint propagation algorithm that may lead to multiple control flow paths (if statements, function calls over multiple paths, loops, etc.). In that case, our sink nodes are marked with $\phi$. Sink nodes marked with $\phi$ are treated as possible results that may arise from different paths.

## 4.13. Post-Analysis Module *M13*

The post-analysis module *M13* aims to determine the possibility of violating a cryptographic rule and extract cryptographic primitives from the ASTs. Cryptographic misuse rules are defined in Section 2.3 and detected with the help of the Rules module *M12* and Post Rules module *M11*. Additionally, from the results of static taint analysis, one can determine the cryptographic primitives from the metadata, such as underlying algorithms (ciphers, hash functions), key, IV lengths and many others. A variety of cryptographic functions are addressed; however, not all can possibly be covered due to time limits. A tremendous amount of manual work and implemented scripts are applied using various firmware images as a reference, which are then analyzed manually and later by automatic methods to identify as many cryptographic functions as possible. All the discovered functions are added to the Rules module *M12*, creating the type and the order of arguments as described. In addition, the appropriate post rules are constructed and appended to the Post-rules module *M12*.

### 4.13.1. Abstract Syntax Tree *(AST)*

The Abstract Syntax Tree *(AST)* is created for every tainted source using the taint propagation algorithm until a taint sink is reached. The static analysis has limitations in discovering indirect pointers and indirect function calls that are calculated at run-time. Additionally, any result that is calculated at run-time or based on an input (either user or external input) cannot be discovered. All the limitations of static analysis can be overcome by including a dynamic analysis framework, although, due to time limits, it is left for future work. Furthermore, the analysis cannot be executed if a binary is not supported by *Ghidra's* CPU architectures. The AST is bounded by *Ghidra's* underlying analysis, such as disassembly, function identification, etc. Thus, false positives and false negatives may arise. However, as *Ghidra SRE* tool evolves, our analysis evolves as well.

The *AST* is recursively parsed using Depth First Search (DFS) in order to identify the existence of a constant sink (*taint sink*). All individual P-codes and their produced children are parsed in DFS order to identify all intermediate result nodes. As the tree parsing progresses, basic operations are handled whenever possible. For instance, basic arithmetic operations such as addition, multiplication, and division are solved as the tree is parsed only if their children can be solved to a constant sink. Note that $32$ and $64$ bit architectures are taken into account. Additionally, if a sink is discovered in different execution paths, then all paths are marked as $\phi$ or $\phi$ *const* (if it leads to a constant sink over multiple paths).

In this paragraph, cases that are not covered in the AST parsing are discussed and left for future work. Firstly, the parsing cannot handle standard *C* functions like *strcpy, memset, memcpy* etc., where the results can propagate through the functions' arguments and eventually will result in a constant sink. The same applies to cryptographic functions such as *MD-5, SHA-256* where propagation is not captured. The accuracy of the AST parsing is bounded by the accuracy of the module producing the AST. If operations are not well-defined, false positives and false negatives may also occur. The AST parsing is just a parser and not a solver of any kind, trying to determine if a path will lead to a constant sink or not. Overall, a manual audit of the produced ASTs may overcome many of the issues mentioned above, and those issues are left for future improvements.

A real example of the static taint analysis using a binary in one of the analyzed firmware images is shown below. The C pseudo-code of a few functions is represented in code listing 4.1.

```c
// function that sets the key and perform a decryption using AES
void FUN_0040108c(uchar *param_1,uint param_2,uchar *param_3,uchar *param_4,uchar *param_5){
  AES_KEY key;  // output key stack variable
  // AES set the decrypt key param_3 of 128 bits and save it to key
  AES_set_decrypt_key(param_3,0x80,&key);
  /* code */
  ...
}

// wrapper function
void FUN_00402554(uchar* param_1){

  // function call with a violation of rule S1
  FUN_0040108c(PTR_DAT_004130a0 + 0x20,0x10,PTR_DAT_004130a0 + 0x10,PTR_DAT_004130a0,param_1)
    ;

  return 0;
```

```
}

// wrappers
void FUN_00401780(...){
  /* code */
  ...
  FUN_0040108c(...)
  /* code */
  ...
}
```

Listing 4.1: Real-world example of binary's firmware image vulnerable pseudo-code.

The marked **taint source** is the `AES_set_decrypt_key` function arguments where the prototype is given below as:

```
int AES_set_decrypt_key (const unsigned char *userKey, const int bits,
AES_KEY *key);
```

The **tainted source arguments** and the given types are briefly explained below:

- 1st argument: `userKey` argument is marked as type `bytes`. In order to resolve it, the analysis needs to know how many bytes of data need to be extracted. Thus, the second argument provides us with the key length in bits. A priority is defined where the key length needs to be resolved first.
- 2nd argument: `bits` argument is marked as type `bit`. Priority `2<1` states that 2nd argument needs to be resolved before 1st argument. The integer constant is converted to bytes when the taint sink is found.
- 3rd argument: `key` argument is marked as type `output`. This indicates which function will later use the key.

The **taint propagation** and hence the Abstract Syntax Trees (ASTs) are depicted in Figure 4.3 for the first two arguments. The **taint sinks** are found to be a constant value of $0x80$ in the 2nd argument ($128$ bits, $16$ bytes) of AES key length, and the decryption key to be constant in a memory section ('.rodata' section) pointed at $@0x402e58 + 0x10 = 0x402e68$. Thus, $16$ bytes are extracted from the resolved pointer, leading to a constant key converted in base64. Therefore, in the Post-Analysis module *M13* (Section 4.13), the discovered results are treated as a cryptographic misuse rule, specifically as a discovered constant key for symmetric key cryptography that violates rule *S1* from Section 2.3.

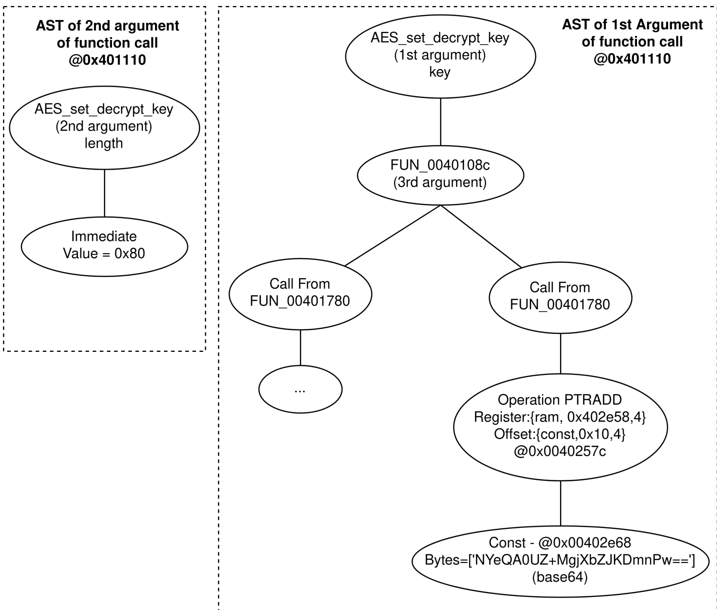



Figure 4.3: Real-world Abstract Syntax Trees (ASTs) for two arguments.

## 4.14. Post Rules module *M12*

The Post Rules module *M12* uses metadata in order to enhance the discovery of a taint sink (function or function argument) to a possible cryptographic violation[10]. Furthermore, it consists of metadata for the discovery of cryptographic primitives. Appendix B describes the variety of supported cryptographic primitives that may be discovered with `EBAT`. Specifically, for the Symmetric Key Cryptography primitives, block ciphers, stream ciphers, mode of operations, key sizes, IV sizes and direction of encryption are discovered. For the Public Key Cryptography primitives, the analysis can identify RSA encryption padding schemes, RSA key sizes, the underlying one-way hash functions for X.509 and digital signatures. Additionally, the analysis can identify random functions from libc, OpenSSL and/or GnuPG libraries. Furthermore, for the Cryptographic One-way Hash function primitives, numerous hash algorithms are identified, e.g., MD-5, SHA-1, SHA-256, etc. For Key Derivation Functions (KDFs) and Password Hashes primitives, a variety of algorithms such as PBKDF1, SCRYPT, crypt (Linux), etc., are covered together with the underlying hash functions, the number of iterations used and the salt. In addition, the Message Authentication Codes (MACs) primitives can identify the HMAC algorithm key size and the underlying hash function. Lastly, for the Authenticated Encryption with associated data primitive, the analysis can cover the underlying algorithm AES and/or CHACHA20 (stream) with modes of Operations like GCM, CCM, etc; in addition, it can determine the key and IV sizes.

As an example of a cryptographic misuse rule, using the implemented post rules, consider the following function prototype from OpenSSL, which performs an RSA encryption using a public key. The last argument denotes the RSA encryption padding scheme that is used.

```
int RSA_public_encrypt(int flen, const unsigned char *from,
unsigned char *to, RSA *rsa, int padding);
```

In order to identify the correct padding scheme and if it is violated or not, for this specific example, post rules must hold all the available padding schemes in the form of `padding = integer`. For instance, in OpenSSL [14], the RSA encryption padding schemes for the 5th argument of the RSA function above are mapped as follows:

- `RSA_PKCS1_PADDING = 1`
- `RSA_SSLV23_PADDING = 2`
- `RSA_NO_PADDING = 3`
- `RSA_PKCS1_OAEP_PADDING = 4`

Weak RSA encryption padding schemes are additionally saved inside the post-rules definition arguments. If the 5th argument (`padding`) is equal to a weak padding such as `RSA_PKCS1_PADDING`, then the rule *P1* is marked as violated (see section 2.3). Otherwise, the discovered padding is saved to the database together with the purpose of having a complete list of used RSA padding schemes for each firmware image. This is not limited only to RSA padding schemes but applies to all the aforementioned cryptographic primitives and all the functions covered in the Rules module *M11*, where the metadata or a result of a function is meaningful. For instance, all the metadata of this OpenSSL `EVP_aes_256_cbc()` function call (AES, 256 bits with CBC mode of operation) are saved from the Post Rules module to the database.

## 4.15. Results Database Module *M14* and Meta-Results Analysis

A database is created for every product that consists of several firmware images. The results of all modules, as described in previous sections, are saved. Briefly, the database for each product containing one to multiple firmware images consists of a product name, type, vendor and all firmwares names, versions, and release dates. In addition, all the results from fuzzy hashing, binary hardening, CVEs, cryptographic library, credentials, post-analysis module and numerous others are stored inside the database. Individual files depending on `EBAT's` user arguments are also stored for future and manual reference.

The first step of meta-results analysis is to merge all databases for each product into a larger one for each vendor. Subsequently, the following is calculated:

[10] The complete list of post rules can be found at EBAT-public post rules, `https://github.com/ppanagiotou/EBAT-public/blob/master/configurations/postrules.cfg`

1. Map every found cryptographic library version to their release date. The release dates for each library are harvested from their respective public websites.

2. Calculate an estimation of similarity for each product using Algorithm 2. It calculates an estimation (percentage) of similarity representing how much of the developed code is similar compared to every other older firmware release. This metric will be used later in Chapter 5.2 when presenting the results and findings.

**Algorithm 2:** Calculate the percentage of code updates for each product.

```
Result: Percentage for each product
foreach product do
    currentFirmware = next Firmware order by release date
    foreach older firmwares do
        foreach binary do
            if binary.digest == currentFirmware.binaries.digest then
                percentage = 100
            else
                percentage = max{∀ binaries compare fuzzy hashing}
            end
        end
    end
end
```

During `EBAT's` analysis, various password hashes are discovered from *Linux 'passwd, shadow'* files. In the Meta-Results Analysis module, an effort is made to crack those passwords using Graphical Processor Units (GPUs). The main tool used is hashcat [8], a powerful password recovery tool with numerous built-in attacks for a variety of hashes. It uses *OpenCL or CUDA* to optimize the performance of cracking thousands or even millions of hashes per second (H/s). The attacks depend on the difficulty of the underlying hashing algorithm and whenever a *unique salt* is used. A few attacks that are used are brute force, dictionary, mask and rule-based attacks for extending our dictionary (provided by hashcat). The findings and the results are presented in Chapter 5, Section 5.4.1. Overall, using the databases created and merged one for each vendor, in Chapter 5, the results and findings of our thesis are presented.

# 5 Results & Findings

This chapter contains a comprehensive evaluation of the implemented tool called *E*mbedded *B*inary *A*nalysis *T*ool (EBAT) which is provided open source[1]. Our evaluation is performed on a dataset containing more than $36,000$ firmware images belonging to more than $5,000$ different products, harvested from $33$ vendors in a date span of over $20$ years. The products are categorized as Internet of Things (IoT) and Embedded devices that are used either in personal or corporate environments, such as routers, security cameras, smart plugs, etc. The large-scale analysis is performed only on the firmware images of each product across their releases. No devices are involved, and no intrusive online testing of any kind was performed, thus making the analysis scalable. In this work, the term 'product' refers to a vendor's product, e.g., a smart sensor. Each product may have one or more firmware images released by the vendor. Firmware images are gathered from publicly available sources and organized in our evaluation corpus, spanning from their initial release to the latest crawled version.

The rest of this chapter starts with the evaluated corpus presented in Section 5.1 where our complete dataset of firmware images will be presented along with the numbers of successfully unpacked firmware images across all vendors (more than $60\%$). Furthermore, we show that *ARM and MIPS* are the most prevailed CPU architecture in *IoT/embedded* industry. Section 5.2 presents our findings on the frequency of device updates. Additionally, a comparison of identical binaries is presented between all vendors, revealing a significant percentage of similar binaries across different vendors' firmware images. In Section 5.3, our results and findings for the indications of exploit mitigation techniques found on firmware's binaries are presented. The lack of exploit mitigation techniques in *IoT/embedded* firmware images is noteworthy. Section 5.4 and 5.4.1 presents the results and findings of all the discovered credentials, mainly focused on private credentials such as private keys and also password hashes that are found embedded in a firmware image. This section provides two real-world case studies showing the significance of private keys discovered in firmware images. Furthermore, Section 5.5 illustrates the discovered open-source cryptographic libraries, including an analysis of their version found embedded in firmware images. Following Section 5.6 provides an analysis of the discovered CVEs affecting the cryptographic libraries.

In Section 5.7, we present the results of our static taint analysis using EBAT to detect cryptographic misuses in firmware image binaries utilizing API calls from well-known cryptographic libraries. The analysis is based on $18$ cryptographic misuse rules as given in Chapter 2.3. In total, the taint analysis for detecting cryptographic violations is executed on over $1.4$ million binaries belonging to $22,548$ successfully unpacked firmware images. Our total evaluation results indicate that over $10$ thousand firmware images are found to be violating at least one of the $18$ rules (except rules ***R1*** and ***R2***), concluding a violation in more than $50\%$ of the examined images. However, no violations of the rules above (except ***R1*** and ***R2***) are discovered for $10,885$ $(48.27\%)$ and $12,040$ $(53.40\%)$ for the case of 'entry and possible $\phi$ node' and for the case of 'entry and not discovered $\phi$ node', respectively. Additionally, various case studies on real-world vulnerabilities in firmware images are presented, including recent CVEs that are found in various vendors' products and executing EBAT on those vulnerable firmware images; we test the effectiveness of our tool to evaluate the automatic capturing of these known vulnerabilities

[1] EBAT is provided open source at EBAT-public, `https://github.com/ppanagiotou/EBAT-public`

or not. In addition, performing large-scale analysis on an extensive corpus of firmware images allows us to discover that other firmware images are affected by these known vulnerabilities, in some cases also from various product lines, that are not covered on the public CVEs reports.

## 5.1. Evaluation Corpus

In order to evaluate our implemented tool, a large-scale analysis is conducted for 33 vendors, including more than 5,000 different products containing more than 36,000 firmware images in a date span of over 20 years. Our evaluation dataset contains firmware images of products from various categories, including those for personal and corporate use. Some of these are VPN Routers, 4G-3G Routers, DSL-Routers, Access Point Routers, Powerlines, Smart Plugs, WiFi Extenders, Security Cameras, IP Cameras, NAS, Cloud Camera Recorders, Switches, PoE Switches, Smart Switches, Firewalls, WiFi Motion Sensors, Satellite Networks Routers, Bridges and Mesh WiFi Systems.

The evaluated dataset consists of firmware images that are captured only from publicly available sources using mostly a crawler, as explained with more details in Chapter 4.1. Each vendor's product may consist of one or multiple firmware images representing subsequent updates from the initial release. Along with the firmware images, the release dates are also captured and saved to the dataset database. Vendors may choose to deploy exactly the same firmware image across different product brands. Thus, those firmware images are merged into one to avoid reanalyzing them. In total, `EBAT` successfully unpacked 22,548 (including 424 partially unpacked firmware image) from 36,073 firmware images, achieving a percentage of over $60\%$[2] from 5,853 unique products across 33 vendors in a date span of almost 24 years.

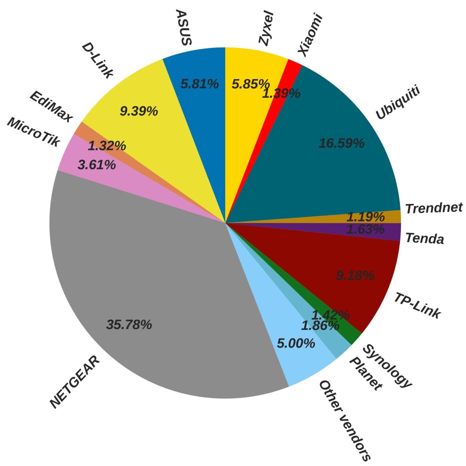



Figure 5.1: Overall distribution of unpacked and partially unpacked firmware images.

Table 5.1 depicts the dataset statistics from the top 10 vendors that have the most unpacked firmware images using `EBAT's` firmware unpack module analyzed in Chapter 4.2. The first column in Table 5.1 serves as a cross-reference to the corresponding table in Appendix A, Table A.4. Appendix A, Table A.4 provides the complete dataset statistics table for the 33 evaluated vendors, including the dates of the oldest and newest firmware images captured per vendor. Each product may consist of one to many firmware images ordered by their release dates and represented as a *5-tuple* element `(Vendor, Product Name, Product Type, Firmware images, Release dates)`, which is stored in the dataset database. A firmware image is marked as successfully unpacked if at least one binary is found during the extraction process and all required dynamic libraries are present. Conversely, a firmware image is marked as partially unpacked if at least one binary is found, but some required dynamic libraries are not present. Unpacked products are categorized as the ones that may have at least one firmware image successfully unpacked or partially unpacked.

Overall, Figure 5.1 illustrates the distribution of unpacked and partially unpacked firmware images over all vendors. Table 5.2 illustrates the various discovered CPU architectures from the unpacked firmware images, including the partial ones. The table also includes information about the bit architecture and endianness. We can interpret the table as follows: $28.46\%$ of the firmware images use the ARM architecture, with $98.77\%$ of this subset using a 32-bit architecture, and within this, $2.62\%$ are big-endian (BE). Additionally, Appendix A Table A.3 depicts, for each vendor separately, a complete list of the chosen CPU architecture for their products.

In our dataset, the firmware images are mostly Linux-based embedded that consist (but are not limited to) of a variant of the Linux kernel, a set of open-source software packages and a set of custom vendor-developed applications [90]. The dataset consists of firmware images spanning over a period of more than 20 years, starting from the oldest one released on *22/04/1997* to the latest capture at the time of writing, released on *28/04/2021*. `EBAT` successfully unpacked over $60\%$ of the captured firmware images. *NETGEAR* has the most unpacked firmware images followed by *Ubiquiti*, *TP-Link*

[2]The unpacked firmware images percentage is calculated over every capture firmware images including the duplicate one that is found in different products.

| # | Vendor | # Prod. | # Firm. | # unique Firmwares | # unpack Products[3] | # partially unpack[4] | # unpack [5] | Total |
|---|---|---|---|---|---|---|---|---|
| 18 | NETGEAR | 829 | 9,458 | 3,790 (40.07%) | 553 (66.71%) | 194 | 7,867 | 8,061 (85.23%) |
| 30 | Ubiquiti | 253 | 3,773 | 935 (24.78%) | 249 (98.42%) | 0 | 3,737 | 3,737 (99.05%) |
| 24 | TP-Link | 950 | 3,258 | 3,210 (98.53%) | 640 (67.37%) | 11 | 2,058 | 2,069 (63.51%) |
| 9 | D-Link | 789 | 3,861 | 3,333 (86.32%) | 359 (45.50%) | 62 | 2,054 | 2,116 (54.80%) |
| 33 | Zyxel | 515 | 2,849 | 2,825 (99.16%) | 231 (44.85%) | 20 | 1,297 | 1,317 (46.23%) |
| 1 | ASUS | 265 | 1,515 | 1,468 (96.90%) | 220 (83.02%) | 42 | 1,267 | 1,309 (86.40%) |
| 17 | MicroTik | 20 | 826 | 826 (100.00%) | 9 (45.00%) | 0 | 814 | 814 (98.55%) |
| 20 | Planet | 290 | 816 | 718 (87.99%) | 162 (55.86%) | 13 | 405 | 418 (51.23%) |
| 25 | Tenda | 267 | 707 | 699 (98.87%) | 135 (50.56%) | 4 | 363 | 367 (51.91%) |
| 23 | Synology | 61 | 319 | 318 (99.69%) | 61 (100.00%) | 0 | 319 | 319 (100.00%) |
| - | **Total** | 5,853 | 36,073 | 24,366 (67.55%) | 3,413 (58.31%) | 424 | 22,124 | 22,548 (62.51%) |

Table 5.1: Top 10 vendors order by dominant unpacked firmware images (Complete results are located at Appendix A Table A.4).

| ARM | | | | MIPS | | | | Other architectures | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 28.46% | | | | 51.32% | | | | 20.22% | | | |
| 32 bit | | 64 bit | | 32 bit | | 64 bit | | 32 bit | | 64 bit | |
| 98.77% | | 1.23% | | 95.77% | | 4.23% | | 68.66% | | 31.34% | |
| LE | BE | LE | BE | LE | BE | LE | BE | LE | BE | LE | BE |
| 97.38% | 2.62% | 100% | 0.00% | 33.44% | 66.56% | 0.00% | 100% | 60.01% | 39.99% | 0.00% | 100% |

Table 5.2: Various CPU architectures for all successfully unpacked and partially unpacked firmware images.

and *D-Link*. Additionally, many vendors deploy identical firmware images for multiple products as only $24,366$ firmware images are unique over $36,073$, meaning that a high percentage of identically firmware images ($33\%$) is also used in different products. *ARM* and *MIPS* CPU architectures are the ones that prevailed the *IoT/embedded* industry mainly on *32-bit* processors with a percentage of over $75\%$.

### 5.1.1. Validity of results

The evaluation dataset consists of a plethora of *IoT/embedded* firmware images, most of them Linux-based with a range of over $20$ years. `EBAT` manage to successfully unpack a percentage of over $60\%$ of those, that using the other modules from our tool, we will manage to perform deeper analysis on the successfully unpacked firmware images (a unique set of $24,366$ firmware images). Most unpacked firmware images use the *ARM* and *MIPS* CPU architecture. The unpacking module tries to be as thorough as possible. However, some firmware images cannot unpacked due to obfuscation or encryption. For the ones that we successfully unpacked, there is a slight possibility that the unpacking procedure is not fully complete, where there is a possibility of missing or corrupted files. On the other hand, our metric of capturing the binaries and finding the presence of all dynamically linked libraries verifies with high confidence that the unpacking progress is as complete as possible.

The evaluation dataset tries to be as complete as possible regarding a product's firmware images and their release dates. There are cases in which specific vendors may not publicly make all the previous releases of a product's firmware images available. Although we tried to overcome this by crawling specific vendors within a year to collect as many 'final' (at each time of crawled) releases of firmware images as possible, the firmware images crawled for a product, spanning from the initial release to the latest crawled version, may not be complete, and subsequent releases may be missing. Additionally, the release date of each firmware image was also an intricate part of discovering, and a procedure as described in Chapter 4.1 is followed, where, in a few cases, a false date might exist in our evaluation dataset.

## 5.2. Firmware Update

In this section, we want to investigate how often the *IoT/embedded products* are updated, what parts of their firmware are regularly updated and in what percentage. All the information that is provided in this section is extracted only from publicly available sources. A firmware update may also exist that is not publicly available. The outline of this section is the following: First, the results (Figures and Tables) are presented and explained. Lastly, conclusions, along with the validity of the results, are presented.

We want to investigate how regularly each vendor updates their products. Figure 5.2 depicts a letter-value plot (The Boxplot for Large Datasets [50]) that shows how often a firmware update occurs for vendors that have the most firmware updates in our dataset (dominant vendors). Additionally, in these types of figures, a horizontal line is plotted that depicts the mean value. Generally, a letter-value plot is an advancement of Box Plots that can summarize the distribution of a dataset using recursively defined boxes to visualize the different partitions of a dataset. Appendix Figure A.3 presents the same letter-value plot for all 33 vendors. The presented plots are created as follows: For each vendor's product with more than one firmware image, it is initially sorted by their release dates. A day gap is then calculated, showing how many days have passed from a firmware release to the next one. For instance, assume that product A has the initial firmware image released on 01/01/2020 and the next firmware update is released on 01/01/2021; then, 365 days has passed once the initial release to the next one. All the calculated gaps are then plotted in a letter-value plot as depicted in Figures 5.2 for top 11 vendors and for all vendors and appendix figure A.3 for all vendors in our dataset.

Additionally, Appendix Table A.7 depicts the mean values of the day gap value for each vendor, where these values also include the outliers plotted in the figure mentioned above. For instance, the results show that *ASUS* updates their products on an average of 125 days, and Zyxel has an average of 184 days. MicroTik and QNAP have the lowest mean values of 20 and 34 days, respectively. On the other hand, vendors such as Planet and D-Link have mean update intervals of 366 and 241 days, respectively, spanning more than six months. Overall, there are instances where vendors consistently provide updates, ranging from a few weeks to months, while others may not release a firmware update for as long as a year on certain products.

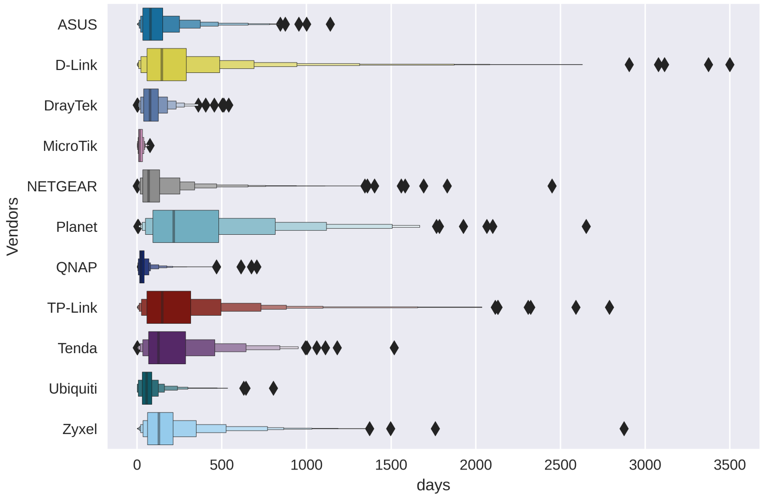



Figure 5.2: Firmware update gap in days over top 11 vendors.

For every successfully unpacked firmware image, a set of binaries is present. A discovered set of binaries is found in most Linux-embedded firmware images that are primarily covered in our dataset. A binary can be categorized into a library or an executable. Appendix Table A.5 presents the total discovered binaries (executable and libraries) found for each vendor's firmware image, where-also the percentage of those that are unique across vendors. In total, EBAT successfully extracted from more than 22,000 firmware images a total of more than 13 million (13,701,913) binaries where approximately 60% are libraries and 40% are executables. Of those binaries, only 989 thousand (989,129) (less than 8%) are unique (using *SHA-256* digest) across vendors' extracted binaries.

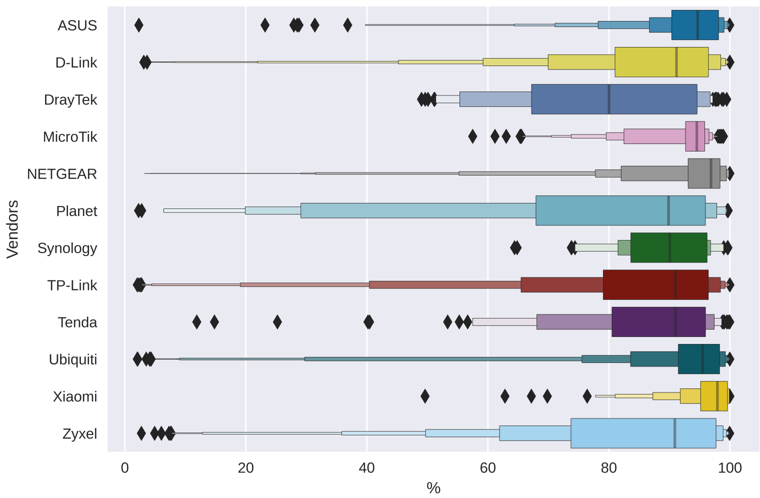
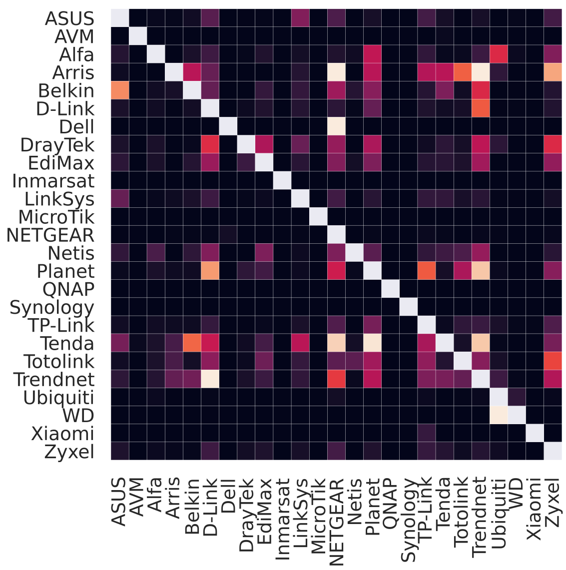



Figure 5.3: Firmware update over all binaries per dominant vendors.

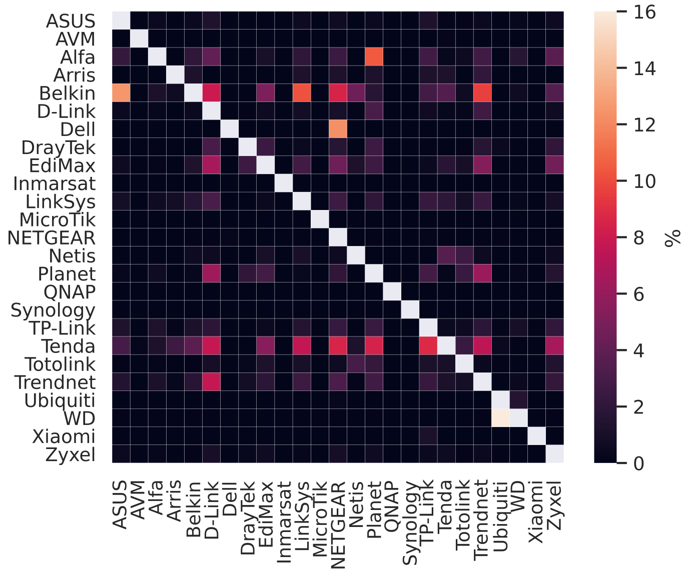
For every binary, the *SHA-256* digest is calculated and saved in our database. Utilizing all the *SHA-256* digests from each binary from each vendor; comparisons are made across different vendors' binaries to calculate the percentage of identical binaries found between each vendor. Thus, heatmaps are plotted where each cell shows a percentage of identical binaries across vendors (normalized). Figure 5.4a illustrates a heatmap across vendors' binaries in each cell, showing the percentage of identical binaries that one vendor has from any other vendor. Furthermore, Figure 5.4b depicts exactly the same but across vendors' binaries that are using only a cryptographic library as analyzed in Chapter

4.6.

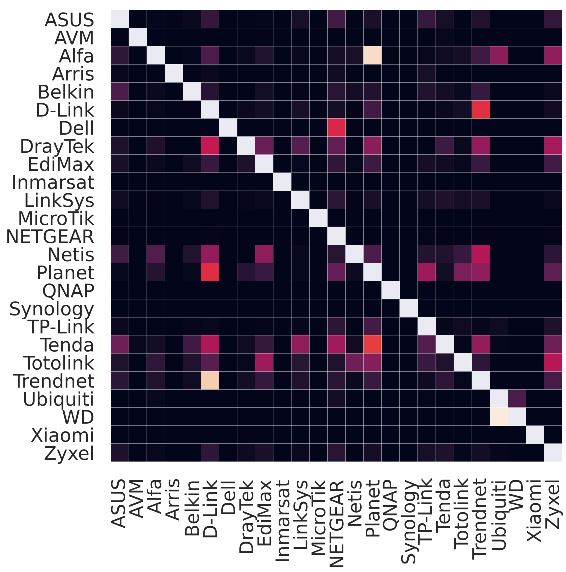
The heatmaps are not symmetrical, as the percentage is calculated based on the total binaries of each vendor represented along the $y$ axes. For instance, *ASUS* has approximately $5.31\%$ identical binaries in common with *Linksys*, while *Linksys* has around $4.32\%$ identical binaries with *ASUS*. In addition, the same calculations are performed individually for executable and library binaries as depicted in Figures 5.4c and 5.4d respectively. Appendix Figures A.1 and A.2 present heatmaps of executable binaries and library binaries that use only a cryptographic library, respectively. Note that some vendors are not plotted in the heatmaps above as the extracted binaries that are successfully found from `EBAT` are below $1000$ and thus are discarded from the plots.



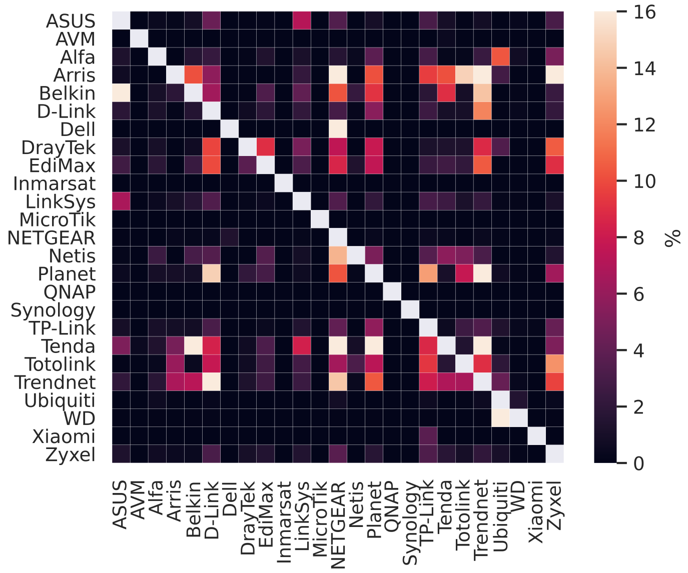

(a) All binaries (executables and libraries).




(b) Binaries that are using a cryptographic library.




(c) Executable binaries.




(d) Library binaries.

Figure 5.4: Heatmaps of duplicate binaries across vendors.

From the aforementioned figures, one can generally observe that executable binaries are less shared across vendors than library binaries. The same also holds in binaries that use a cryptographic library with even greater uniqueness between vendors' binaries. Our results show that not all vendors share binaries with each other; however, a portion of them do. For instance, vendors use identical binaries with *NETGEAR* such as *TP-Link* and *Trendnet* with approximately 3.36% and 9.41% respectively. On the other hand, some vendors use an insignificant percentage of identical binaries between every other vendor, such as *AVM* (only 0.39%).

Generally, each binary EBAT calculates 2 digests. The *SHA-256* digest that shows the uniqueness of a binary and a fuzzing digest or similarity digests is called *ssdeep* [53]. *Ssdeep* can be compared with every other binary, producing a result from $0 - 100$ (percentage) that indicates the degree of similarity between the two binaries. We perform pairwise comparisons between each binary using the algorithm provided in the previous Chapter 4.15, Algorithm 2. Subsequently, the results are aggregated, where a percentage is calculated for each product that consists of more than one firmware image. This percentage shows the similarity among the binaries across different firmware releases; 100% means that all firmware's updated binaries are identical with any of the previous releases, and 0% means that no firmware's binaries are found to be the same with any of the other previous releases. Generally, a firmware image may include other files except binaries, such as credentials, web pages, configuration files, etc. Thus, 100% similarity percentage on binaries is plausible where 0% or near 0% is considered an outlier.

These percentages are calculated for every vendor's product, and letter-value plots are presented. Figure 5.3 (for dominant vendors) and A.4 (for all vendors) depicts the percentage of firmware update similarity calculated for all binaries, where Figure 5.5 (for dominant vendors) and A.5 (for all vendors) shows the percentage of similarity calculated only on 'crypto' binaries (executables and libraries). 'Crypto' binaries are the ones that use one or more cryptographic libraries as described in Chapter 4.6. The mean values for the aforementioned figures for all vendors are given in appendices tables A.8 and A.9 for all binaries and 'crypto' binaries, respectively. Comparing Figure A.4 with A.5, we can generally observe that 'crypto' binaries are updated more frequently than 'non-crypto' binaries.

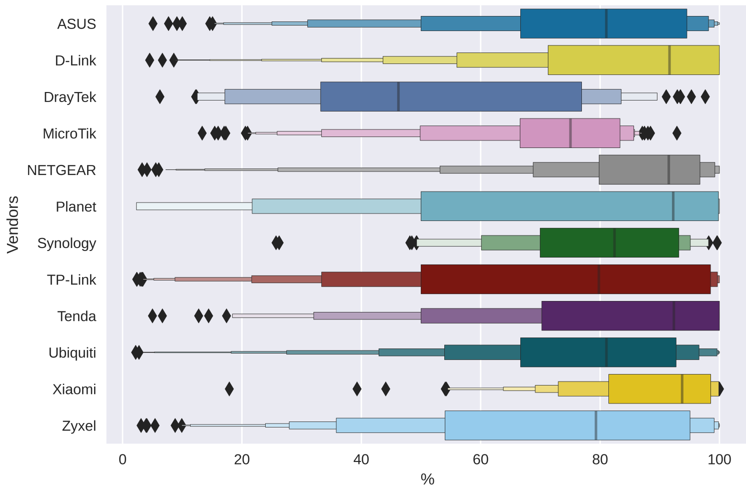



Figure 5.5: Firmware updates for binaries using a dynamically cryptographic library (executables and libraries).

## 5.2.1. Conclusions and Validity of results

The letter-value plots depicted in Figure 5.2 and A.3 suggest that *IoT/embedded* products might not be very consistent in providing regular firmware updates. Those may also include security patches, which are essential for the overall product's security. While the discovered binaries across all vendors are more than 13 million, the unique binaries are only 8% of those (see Appendix Table A.5). This observation leads us to speculate that there is a commonality in the utilization of identical tool-chains, original equipment manufacturers (OEMs) products, software tools, API libraries, and other development resources across various vendors for the development of firmware images. It suggests a trend where similar sets of tools and resources are consistently employed in the firmware development process across a majority of vendors. Heatmap figures provided in the above section reinforce the same speculations mentioned earlier, where one can observe identical binaries (executables and libraries) across multiple vendors. Binaries that use a cryptographic library are less commonly found to be identical across multiple vendors, as depicted in the heatmaps. Furthermore, in the firmware update letter-value plots, we observe that updates in binaries surpass 10% between firmware releases. Specifically, this percentage increases significantly in binaries using a cryptographic library to over 30%. It may indicate that a firmware image's changed/patched security-related features are a priority over other features.

Our primary concern regarding the validity of the above results is the consistency of having all the firmware images, starting from the initial release until the final release (at the time of crawl) across a product. Not all vendors may provide this information, and despite the efforts that are made to minimize this, some products may consist of an incomplete set of firmware updates, as well as their release dates. Additionally, firmware updates may be applied to a product without being released publicly. As a consequence, missed firmware images and their release dates may provide false positives in our results regarding the update date span. Finally, partially or even successfully unpacked firmware images may miss a few binaries from successful extraction, thus limiting the true positives on our binaries' results.

## 5.3. Exploit mitigation techniques on firmware images

The exploit mitigation techniques (hardening security features) are analyzed in Chapters 2.2 and 4.4 are used to prevent mainly memory corruption bugs in binaries. These techniques' absence or limited presence weakens overall system security, increasing the feasibility of creating an exploit when a memory corruption bug is discovered. `EBAT` can discover indications of these techniques (if they may exist or not) on each binary extracted from a firmware image. In this section, we examine whether the *IoT/embedded* product's firmware images use exploit mitigation techniques on their binaries and in what percentage. For a fair comparison, `EBAT` was also executed on the binaries of a state-of-the-art system, the base image of the latest ARM-based server (64-bit) Ubuntu Server[6] (at the time of download) and will be compared below.

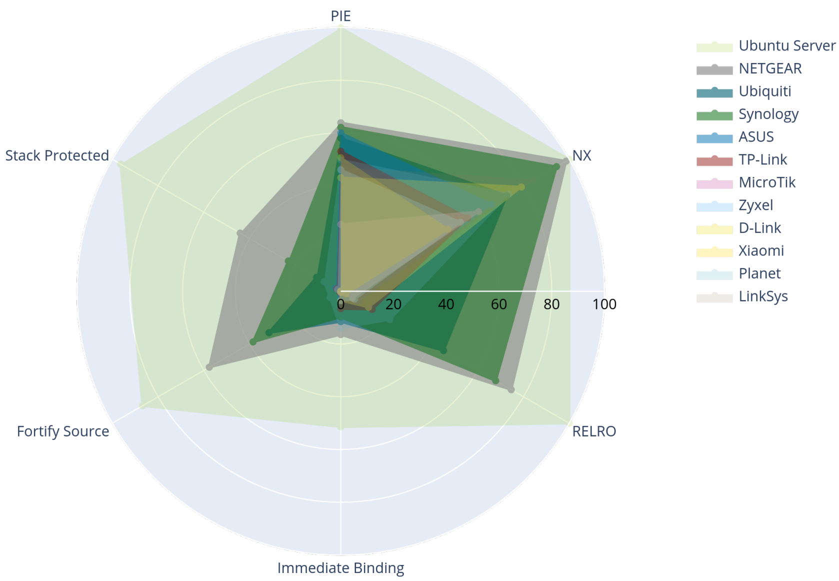



Figure 5.6: Exploit mitigation techniques *IoT/embedded* dominant Vendors versus Ubuntu Server[6].

Figure 5.6 illustrates a radar chart, indicating the percentage-wise presence of exploit mitigation techniques discovered for dominant vendors. All the charts are created only with the binaries that indicate whether an exploit mitigation technique is discovered or not. If an indication is marked as probably exists or not discovered, then the binaries are discarded and marked as *'not found'*. Overall, in Appendix A.4, various charts are presented and grouped by each vendor and CPU architecture. In addition, Appendix Tables A.10, A.11 and A.12 present with more details the results of exploit mitigation techniques for each vendor separately. As a reference point, an analysis is performed on binaries of ARM 64-bit base image of Ubuntu server[6] where the results of these indications of exploit mitigation techniques are also presented in Figure 5.6 for comparison. Figure 5.7 depicts a radar chart for exploit mitigation techniques for the two dominant CPU architectures, ARM and MIPS, both 32 and 64-bit in little and big endian, for dominant vendors compared with the state-of-the-art Ubuntu ARM base image[6].

Generally, a binary should have as many exploit mitigation techniques as possible. Thus, the bigger the area in the aforementioned radar charts, the better. Comparing with the state-of-the-art latest ARM 64-bit base image of Ubuntu server[6] with the dominant vendors, one can observe the lack in *IoT/embedded* products to deploy the exploit mitigation techniques that are present for over a decade. Fortunately, the non-executable bit (*NX*) is almost present with a very high percentage in *IoT/embedded* products' binaries. *PIE* is present in approximately $60\%$ of dominant vendors while stack canaries (*Stack Protected*) varies from a low percentage to $40\%$ in *NETGEAR* binaries. *RELRO* also varies and has a range from almost $4\%$ to nearly $75\%$. In comparison with the state-of-the-art ARM Ubuntu

[6]Ubuntu ARM 64 Base 20.04.2 LTS (Focal Fossa), with CPU architecture `aarch64`, located at https://cdimage.ubuntu.com/ubuntu-base/releases/20.04.1/release/, released on 01/02/2021 (SHA256 - filename)(e5d384385b59b0c1d7103e096034fa962e7d98c23db2b17481f4da55a1613804 - ubuntu-base-20.04.2-base-arm64.tar.gz)

Server[6] where *PIE*, *NX*, *Stack Protected* and *RELRO* are nearly 100% present.

### 5.3.1. Conclusions and Validity of results

The indications of exploit mitigation techniques on firmware images' binaries are presented in this section with interesting findings. *IoT/embedded* product firmware images tend to limit the usage of hardening security features on their binaries, causing the weakening of overall system security when a vulnerability is discovered. The lack of exploit mitigation techniques causes recent binary exploitation attacks to be executed successfully, as shown in Chapter 2.2. Vendors need to further adapt to recent exploit mitigation techniques and implement them in their products, with the ultimate goal of enhancing overall system security.


Figure 5.7: Exploit mitigation techniques ARM vs MIPS vs Ubuntu Server[6] *IoT/embedded*.

Regarding the validity of the aforementioned results, as explained in Chapter 2.2, there is a small probability of false positives in some indications, such as Fortify Source functions. Finally, these are only indications of exploit mitigation techniques discovered in binaries and not verified in any way by obtaining the devices and if the device implements these mitigation techniques correctly. Overall, there are concerns regarding the low usage compared to a state-of-the-art system.

## 5.4. Credentials and Password hashes

This section analyses the extracted information data from every successfully unpacked firmware image, focusing on credentials such as private keys and passwords (mainly in hashed form). Private keys and plaintext passwords pose a security risk when discovered embedded in a firmware image. Additionally, hashed passwords and encrypted private keys may exist in a firmware image. However, password hashes may be purely hashed with outdated algorithms or even non-unique salts, and the encrypted private keys may be using outdated algorithms or the encrypted method, e.g., a password should also be unique, follow the password strength requirements, and not found embedded in a firmware image. In the rest of this section, the results will be presented from our evaluation corpus for various discovered credentials that pose a security risk on the device, e.g., an SSH private key. In addition, password hashes will be presented along with an analysis of discovered passwords (cracked) found using publicly available resources such as dictionaries[7] as well as not publicly available ones (disclosure is not possible).

Table 5.3 depicts the total discovered credentials found in our entire dataset. The credentials are discovered using the credentials module analyzed in the previous Chapter 4.8 that verifies the validity of each credential but not its usage. For example, if an SSH private key is discovered, but the product has not enabled the SSH service to accept an incoming connection, then the key is there but not exploitable to an adversary since the SSH service is disabled. Publicly available credentials such as certificates, public keys, and PGP public keys do not pose any security risk as they are publicly available to anyone. On the other hand, private keys and SSH private keys must remain private and not be discovered in any firmware image, as this will break the overall product's security. In total, a high percentage of 27.98% of the successfully unpacked firmware images hold an unencrypted private key, either found in a separate

[7]Open-Source SecLists Github dictionaries, `https://github.com/danielmiessler/SecLists/tree/master/Passwords`, Commit version 545e57b02d71d5a177c8c5896ed5dca8131580ae, `https://github.com/danielmiessler/SecLists/commit/545e57b02d71d5a177c8c5896ed5dca8131580ae`

file or embedded in a binary.

Additionally, in 975 (4.30%) firmware images, an SSH private key is extracted that is unencrypted as well. The abbreviation *'encrypted'* refers to the password-protected credentials, and *'decrypted'* refers to the encrypted credentials that are successfully decrypted using a *'known'* password discovered by manual audit embedded in firmware images. Due to a lack of time and simplicity, the known discovered passwords list is created based on a few passwords that are either very popular or found by manual analysis that is made on selected firmware images. During this manual analysis, we discovered plaintext passwords on scripts that create and/or secure credentials. The password list that is checked in order to decrypt a credential is the following: `password, whatever, deadbeef, root, root12345, admin, N*****3, T****************3`, where '*' (star symbol) is used to non-disclose any of the passwords that are not publicly available.

| Types | Total | | | Embedded | |
|---|---|---|---|---|---|
| | # Credentials | # Firmwares | % | # Credentials | # Firmwares |
| Certificates | 1,278,943 | 7,981 | 35.39% | 4,020 | 399 |
| Public Keys | 23,271 | 7,572 | 33.58% | 655 | 385 |
| Private Keys (not encrypted) | 14,877 | 6,305 | 27.96% | 367 | 209 |
| Private Keys (encrypted) | 1.182 | 970 | 4.30% | 108 | 108 |
| Private Keys (decrypted) | 1,244 | 393 | 1.74% | - | - |
| Various cryptographic Parameters | 3,549 | 2,203 | 9.77% | - | - |
| Certificate Signing Requests | 972 | 474 | 2.10% | - | - |
| SSH Private Keys (not encrypted) | 1,852 | 975 | 4.32% | 2 | 2 |
| SSH Private Keys (encrypted) | 4,041 | 240 | 1.06% | 2 | 2 |
| SSH Public Keys | 1,872 | 986 | 4.37% | - | - |
| PGP Signatures | 39,986 | 523 | 2.32% | - | - |
| PKCS12 (encrypted) | 90 | 90 | 0.40% | - | - |
| PKCS12 (decrypted) | 692 | 320 | 1.42% | - | - |

Table 5.3: Total discovered credentials over our entire dataset.

### 5.4.1. Password hashes

On every successfully unpacked firmware image, `EBAT` tries to discover hashed passwords found mainly in Linux-based firmware images. Remarkably, at least one hashed password is discovered in 5730 (25.99%) of our firmware images. These are hashes located in `'passwd'` or `'shadow'` files mainly used for user password login. The aggregate results and information will be presented below, along with an effort to crack those hashes using publicly available resources and non-public ones, as explained in Chapter 4.15.

In Table 5.4, the total information of discovered password hashes and the attempt to find the actual password (cracked) is presented. Furthermore, Appendix Table A.14 reveals the information mentioned above for each and every vendor separately, where symbol '✓' counts the number of hashes that are successfully cracked; otherwise, symbol '✗' is used. Similarly, Table 5.5 depicts the type of Unix hashes and how many of those are discovered (cracked) or not where symbol '✓' counts the number of hashes that are successfully cracked; otherwise, symbol '✗' is used. The high usage of outdated hashed algorithms like *'DES-based'*[8] and *'MD-5'* (more than 90%) is raising security concerns. In addition, the usage of publicly available passwords is also very high; a percentage of more than 85% of the total found passwords is from publicly available resources presented in[7], which is very concerning. Overall, the top 10 common discovered passwords are: `1234 (26.87%), <empty> (13.71%), ubnt (10.02%), admin (8.22%), F******p (7.21% - not publicly available), root (5.28%), 5up (4.29%), password (4.23%), l*****g (3.43% - not publicly available), realtek (2.07%)`, where '*' (star symbol) is used to non-disclose any of the passwords that are not publicly available[7].

[8]"Traditional DES-based scheme", Wikipedia, `https://en.wikipedia.org/wiki/Crypt_%28C%29%23Traditional_DES-based_scheme`

| Description | # |
|---|---|
| # Firmware images | 5,730 (25.99%) |
| # Overall hashes found | 8,983 |
| # Cracked hashes | 6,668 (74.23%) |
| # Publicly cracked hashes[7] | 5,733 (85.98%) |
| # Unique hashes | 793 |
| # Unique cracked hashes | 290 (36.57%) |
| # Unique publicly cracked hashes[7] | 252 (86.90%) |
| # Non unique salted hashes | 38 (4.79%) |

Table 5.4: Password hashes overall information.

| Hash types | Total | ✓ | ✗ |
|---|---|---|---|
| DES (Unix) | 2,623 | 2,437 | 186 |
| MD5 (Unix) | 6,037 | 3,922 | 2,115 |
| MD5 (ARP) | 149 | 149 | 0 |
| Blowfish (Unix) | 5 | 4 | 1 |
| SHA256 (Unix) | 33 | 27 | 6 |
| SHA512 (Unix) | 136 | 129 | 7 |
| Total | 8,983 | 6,668 | 2,315 |

Table 5.5: Types of Unix hashes.

### 5.4.2. Case studies

A high severity `CVE-2017-14422`[9] with a base score of 7.5 is found on devices *D-Link DIR-850L REV. A (with firmware through FW114WWb07_h2ab_beta1) and REV. B (with firmware through FW208WWb02)*. `CVE-2017-14422` description states: “the same hard-coded `'/etc/stunnel.key'` private key across different customers’ installations is used, which allows remote attackers to defeat the HTTPS cryptographic protection mechanisms by leveraging the knowledge of this key from another installation.” From the stunnel website[10]:“Stunnel is a proxy designed to add TLS encryption functionality to existing clients and servers without any changes in the programs’ code.” To test the effectiveness of `EBAT` in finding hard-coded credentials, we start a search for the identical `'stunnel.key'` of `CVE-2017-14422` which, in the end, we captured it in our results database. Scanning for the same key file in our *D-Link* results database, to our surprise, we discover an additional 195 firmware images (from 32 different products) ranging from the fourth quarter of 2012 to the third quarter of 2020 having the exact hard-coded `'stunnel.key'`.

| Vendor | # Products | # Firmwares |
|---|---|---|
| Actiontec | 1 | 1 |
| D-Link | 35 | 214 |
| EdiMax | 4 | 6 |
| NETGEAR | 7 | 31 |
| Planet | 2 | 2 |
| QNAP | 2 | 23 |
| Totolink | 7 | 22 |
| Trendnet | 2 | 4 |
| Western-Digital | 1 | 1 |
| Zyxel | 1 | 5 |
| **Total** | 62 | 309 |

Table 5.6: Discovered `stunnel` private keys for all vendors.

| Vendor | # Products | # Firmwares |
|---|---|---|
| ASUS | 48 | 171 |
| D-Link | 27 | 118 |
| LinkSys | 4 | 6 |
| NETGEAR | 42 | 324 |
| Planet | 27 | 47 |
| TP-Link | 3 | 7 |
| Trendnet | 17 | 20 |
| Ubiquiti | 6 | 37 |
| Zyxel | 7 | 22 |
| **Total** | 181 | 752 |

Table 5.7: Discovered zebra configurations that use a hard-coded password.

The next step is to scan for any stunnel private keys in all vendors. Table 5.6 depicts all the stunnel found keys for all vendors either in a *key* or a *pem* file. Remarkably, *D-Link* uses the same tunnel key for 195 discovered firmware images and only 19 different keys for other firmware images. Furthermore, stunnel private keys were also discovered in *NETGEAR’s* firmware images, specifically, 31 firmware images across 7 products. Moreover, in a total of 309 firmware images, a stunnel private key is discovered that affects 62 products, raising many security concerns. Although the stunnel private key is known for multiple devices with specific firmware images, we do not attempt in any way to verify it on a public device and due to lack of time, no local verification is attempted either and left for future work. This thesis will not disclose the exact versions of the firmware images that a stunnel private key is discovered.

A high severity `CVE-2021-21818`[11] with a base score of 7.5 is found on *D-LINK DIR-3040 1.13B03* which is an AC3000-based wireless internet router that can cause a denial of service with a specially crafted network request. The vulnerability affects a Zebra service, which is a routing manager that uses a hard-coded password configuration found in file `'zebra.conf'` reported at [63]. `EBAT` also scans and saves configuration files for each firmware image. We wrote a simple module that scanned all of our results from the database to find *zebra* configuration files with hard-coded passwords. To our surprise, the results are depicted in Table 5.7 where for 181 different products, including 752 firmware images, a similar *zebra* configuration file is discovered that contains a hard-coded password! Not all

[9] NVD - CVE-2017-14422, National Vulnerability Database, 2017, `https://nvd.nist.gov/vuln/detail/CVE-2017-14422`

[10] `https://www.stunnel.org/`

[11] NVD - CVE-2021-21818, National Vulnerability Database, 2021, `https://nvd.nist.gov/vuln/detail/CVE-2021-21818`

products with the specific firmware image version may be vulnerable to the same high-severity CVE due to boot configuration options. In addition, no verification on a physical device is performed. In conclusion, EBAT offers the ability to integrate more modules that might lead to interesting results, as the one previously presented.

### 5.4.3. Conclusions and Validity of results

A significant number of private credentials are discovered embedded in firmware images, amounting to more than 25% of the total successfully unpacked firmware images. One of the four firmware images has a private credential embedded in it. However, the usage of these credentials needs to be examined, as private credentials may be regenerated upon boot or even not used at all. However, the case studies presented in this section are examined and confirmed with the CVE reported. Using EBAT, we also discovered the same credentials on the reported CVEs, which were also discovered in subsequent and different firmware image releases than the reported ones. Overall, a private credential cannot stored in any way in plaintext inside a firmware image and needs to be protected, preferably saved in secure storage. No false positives regarding the validity of credentials exist, as the credentials are verified as a file structure using tools to verify them in a valid/correct structure. The usage of those is unknown.

Password hashes are located embedded in over 25% of firmware images, where more than one password hash can be found in a firmware image. The hash of a password alone does not pose an immediate security risk, although a weak algorithm of the hashed password and the non-existence or non-uniqueness of salt does. *DES-based*[8] and *MD-5* hashed password algorithms were the prevailing ones, with over 90% of the discovered ones using outdated algorithms. Along with the hashed passwords, the usage of unique salts was really low, and our efforts to crack those hashed passwords due to the algorithms and the non-existence of unique salts were easier. Additionally, the same hashed passwords were used repeatedly between different firmware images, with more than 85% of the overall discovered hashed being exact duplicates. Overall, out of 290 cracked passwords, 252 are publicly available passwords[7]. Regarding the validity of the discovered hashed passwords, they are found mainly in 'passwd' or 'shadow' files from Linux-based firmware images and are used, as far as we know, for login access to the devices. There is still a possibility that the passwords may change over the first device boot, in a new firmware update, or from the user's input.

## 5.5. Cryptographic Libraries

Cryptographic libraries play an essential role in the overall firmware security of every device. Many functionalities of a device utilize these libraries to implement security protocols, application features, etc. EBAT has the ability to discover the cryptographic library version for popular libraries with high accuracy as presented in Chapter 4.6. In this section, we investigate only publicly well-known cryptographic libraries and not vendor-specific implementations of cryptographic algorithms. We discover the version of a cryptographic library in each firmware image. Then, post-analysis is executed to map the version number to the release date of each library version as well as the end-of-life (*EoL*) date (if it is available at the time of producing the results). In rare cases, a cryptographic library version cannot be identified successfully, maybe due to the stripped library version or different compilation parameters. Those cases are left for future improvement. In the given section, comparisons are performed between the release date of a firmware image and the release date of the discovered cryptographic library version. In addition, aggregate results are presented with discovered end-of-life (*EoL*) of used cryptographic libraries as well as outdated libraries that are used until the latest crawled firmware image release date.

Table 5.8 depicts the total number of discovered cryptographic libraries for every successfully unpacked firmware image, along with the success rate of finding the particular library version. Additionally, for cryptographic libraries that EBAT has discovered the cryptographic library version, we map the release date to the *EoL* date of the given major version. Together with the firmware image release date, we count the number of firmware images with a cryptographic library that has reached the *EoL* date even **earlier** than the firmware image release date. The results are aggregated and presented in the following table as the number of '*End of Life (# EoL)*' that counts the number of firmware images that have at least one *EoL* cryptographic library. In Appendix Tables A.26, A.27, and A.28, the discovered cryptographic libraries results are presented for vendors individually, furthermore, in Appendix Tables

A.29 and A.30, the discovered firmware images that are using an *EoL* cryptographic library **earlier** than the firmware image is released are presented for vendors separately.

| | Libraries | # firmwares | % | # versions | % | # EoL | % |
|---|---|---|---|---|---|---|---|
| 1 | Crypto++ [3] (libcrypto++, libcryptopp) | 4,189 | 18.59% | - | - | - | - |
| 2 | GnuPG [6] (libgcrypt) | 7,056 | 31.32% | 7,056 | 100.00% | 3,943 | 55.88% |
| 3 | GnuTLS [7] (libgnutls) | 3,795 | 16.84% | 3,795 | 100.00% | 403 | 10.62% |
| 4 | KerberosV5 [93] (libk5crypto) | 5,829 | 25.87% | - | - | - | - |
| 5 | Libc (uClibc-ng [16] or glibc [5]) [48] (libcrypt) | 21,266 | 94.39% | - | - | - | - |
| 6 | Libsodium [9] (libsodium) | 205 | 0.91% | 205 | 100.00% | 0 | 0 |
| 7 | LibTomCrypt [10] (libtomcrypt) | 48 | 0.21% | - | | - | - |
| 8 | mbedTLS/PolarSSL [11] (libmbedcrypto, libmbedtls, libpolarssl, libmbedx509) | 859 | 3.81% | 504 | 58.67% | 256 | 50.79% |
| 9 | Mcrypt [12] (libmcrypt) | 700 | 3.11% | 646 | 92.29% | 0 | 0 |
| 10 | Nettle [13] (libnettle) | 3,713 | 16.48% | 157 | 4.23% | 0 | 0 |
| 11 | *OpenSSL* [14] (libcrypto, libssl) | 17,540 | 77.85% | 16,882 | 96.25% | 7,134 | 42.26% |
| 12 | WolfSSL [17] (libwolfssl, libcyassl) | 1,613 | 7.16% | 287 | 17.79% | 186 | 64.81% |

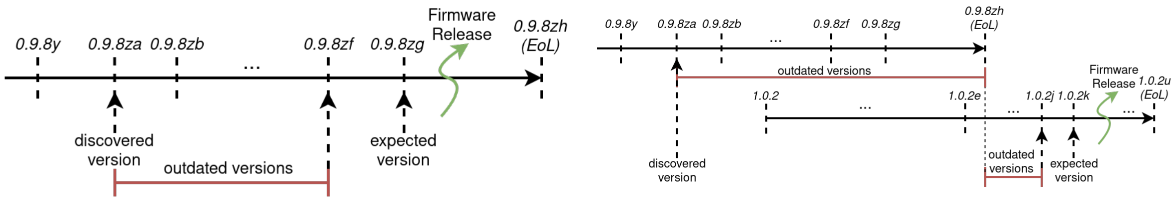
Table 5.8: Discover Cryptographic Libraries over every successfully unpacked firmware image and count the firmware images with at least one discovered *EoL* cryptographic library.

The *libcrypt* cryptographic library from Libc (uClibc-ng [16] or glibc [5])[48] is the most dominantly used library in our dataset of firmware images as discovered more than $94\%$ of the overall successfully unpacked firmware images. The second dominant one is *OpenSSL* [14] (libcrypto, libssl), which is found in nearly $78\%$ of the total successfully unpacked firmware images, and EBAT successfully discovered the cryptographic library version in $96.25\%$ of the discovered firmware images that use the *OpenSSL* cryptographic library. GnuPG [6] (libgcrypt) is used in approximately $31\%$ with $100\%$ version discovery. The usage of other cryptographic libraries follows with lower percentages. It should be noted that a firmware image can consist of one to many cryptographic libraries. Thus, in our results, we count the existence of each one separately per firmware image. Furthermore, we can observe very high percentages of firmware images that use an *EoL* library even **earlier** than the release date of the firmware image. More than $50\%$ of the discovered cryptographic libraries of *GnuPG*, *mbedTLS*, and *WolfSSL* are using an *EoL* outdated cryptographic library, even **earlier** than the publicly released date of a firmware image. *OpenSSL* cryptographic library comes with a lower percentage of approximately $42\%$, which is still very high as the usage of this library is broader. The following section will present an in-depth analysis of the two dominant cryptographic libraries for which a version is discovered: *OpenSSL* and *GnuPG*.

### 5.5.1. OpenSSL and GnuPG cryptographic libraries in firmware images

*OpenSSL* and *GnuPG* cryptographic libraries are broadly used in firmware images, and versions of them have been successfully discovered in $16,882$ and $7,056$ successfully unpacked firmware images, respectively. The libraries mentioned above are the most dominant ones (except *libcrypt* from Libc). In this section, an in-depth analysis of the results is presented. Appendix Tables A.31 and A.32 depict the usage of *OpenSSL* and GnuPG cryptographic libraries on binaries, respectively. A total of $1,452,039$ binaries discovered in firmware images are using an open-source cryptographic library analyzed by EBAT, where $45.5\%$ and $2.99\%$ of those binaries are using the *OpenSSL* and *GnuPG* cryptographic libraries, respectively, as shown in the appendix tables. Almost half of our discovered binaries use the *OpenSSL* cryptographic library, and the others follow with much lower percentages. In the rest of this section, we will investigate the discovered library version that comes with the firmware image, whether it is outdated, and for how long.

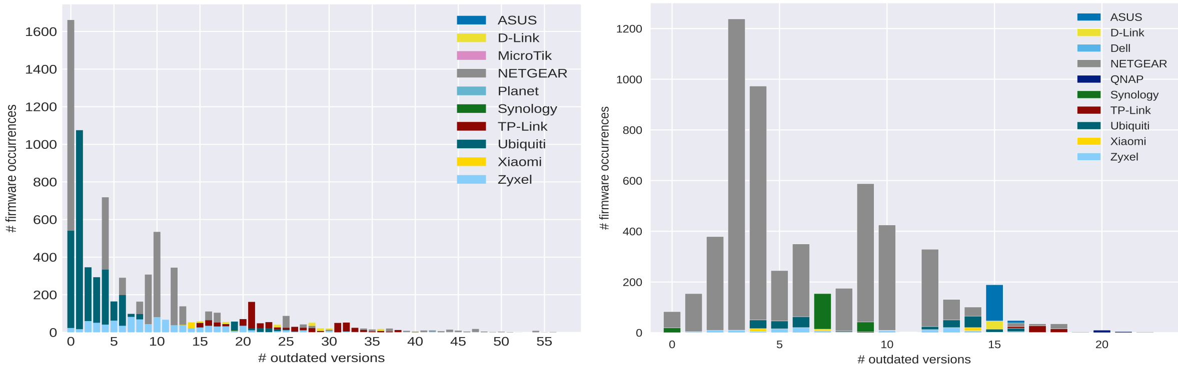
Having the library version information, we map it to their release date and *EoL* date (if they are available when producing the results). We then calculate the gap of outdated versions between the expected cryptographic library version (the latest one released before the public firmware image release) with the discovered cryptographic library. The gap is calculated in releases, meaning that if the release is $0$, then no latest library version exists by the time of releasing the firmware image, and if the release is $x$, then $x$ more recent library versions exist. Figures 5.8a and 5.8b depict examples of how the gap of an outdated version of a cryptographic library is calculated for a firmware image. In the first scenario, assume that a firmware image is released at a given time and EBAT has successfully




(a) Scenario 1 of calculating the gap of outdated versions.

(b) Scenario 2 of calculating the gap of outdated versions.

Figure 5.8: Scenarios of calculating the gap of outdated versions.




(a) Histogram of *OpenSSL* outdated versions.

(b) Histogram of *GnuPG* outdated versions.

Figure 5.9: Histograms of outdated versions.

discovered the cryptographic library version of *OpenSSL* to be *0.9.8za*. The expected cryptographic library version of the given firmware image should be *0.9.8zg*, the latest one before the release time of the firmware image. The gap of outdated versions is calculated to be $6$, which equals the libraries from *0.9.8za* to *0.9.8zf* due to the expected library version of *0.9.8zg*. Scenario $2$ calculated the same but with an outdated version already reaching the *EoL*. In both cases, the red lines present the gap between outdated versions. The aggregate results of outdated versions for dominant vendors are presented as a histogram in Figures 5.9a and 5.9b for *OpenSSL* and *GnuPG*, respectively. Appendix Tables A.18 and A.19 present the results for every vendor separately, where each cell counts the number of firmware images, and each column indicates the number of outdated cryptographic libraries. For instance, in *OpenSSL* table A.18, we have a total of $2,629$ firmware images to be the expected *OpenSSL* cryptographic version and $1,808$ firmwares to be one version behind the latest expected one. Overall in Appendix tables A.18 (*OpenSSL*), A.19 (GnuPG), A.20 (GnuTLS), A.21 (Libsodium), A.25 (mbedTLS), A.24 (Mcrypt), A.23 (Nettle) and A.22 (WolfSSL), the results for every cryptographic library that a version is discovered, are given.

The *OpenSSL* cryptographic library was found to be the latest version ($0$ outdated versions), relative to the public release date of the analyzed firmware image, in $2,629$ from $16,882$ firmware images according to Appendix table A.18. For $1$ outdated version, `EBAT` discovered $1,808$ firmware images, which are still close to the particular library's expected version ($0$ outdated versions). In total, from $0-5$ outdated versions, $8,447$ firmware images have been discovered, and the rest remain at $8,435$, a high percentage of nearly $50\%$. For *GnuPG*, `EBAT` discovers only $121$ firmware images to have the expected *libgcrypt* version. In total, $3,373$ firmware images are found to be using from $0-5$ outdated versions, and the rest of the firmwares ($3.683$ a percentage of over $50\%$) is using a version greater than $5$. Figures 5.9a and 5.9b present the aggregate results of the appendices mentioned above tables, which reveals that many firmware images are using outdated versions greater than $5$, where it may have implications of the overall device security.

With the calculated outdated gap of a cryptographic library version, identical calculations are performed as previously, but with the gap in days (as a time-gap), with an example depicted in Figure 5.10. The time-gap is calculated in days between the expected latest release version *0.9.8zg*, released on *11/06/2015* and the actually discovered version of *0.9.8za*, released on

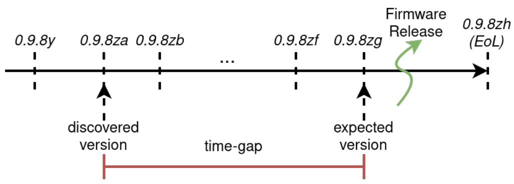
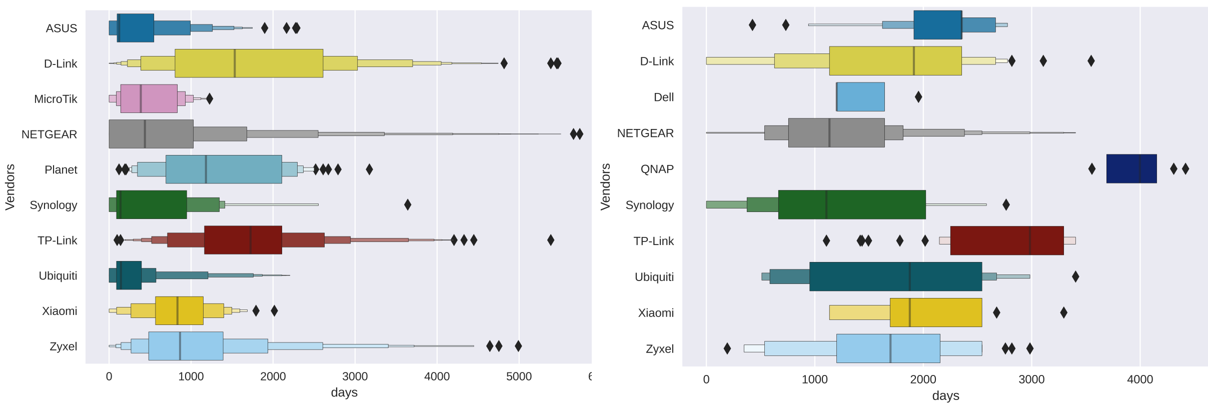



Figure 5.10: Scenario time-gap of outdated versions.

*05/06/2014*, which is more than 365 days old. The results are presented aggregated in letter-value plots for dominant vendors that use the *OpenSSL* and *GnuPG* in Figures 5.11a and 5.11b, respectively. We can observe from those figures that a significant number of firmware images have more than a year-old cryptographic library in their released firmware images. Furthermore, in Appendix Figures A.9 and A.10, the results are presented for every vendor in our dataset, excluding the ones that have no cryptographic libraries discovered. The mean values of the aforementioned plots are given in Appendix tables A.16 and A.17 for *OpenSSL* and *GnuPG*, respectively. In addition, for the outdated versions of the time-gap scenario, Figures 5.11a and 5.11b, a mean value of $x$ in days is calculated, implying that on average a firmware image is released with an outdated version of $x$ days old. These mean values are extracted for the figures above and presented in the aforementioned appendices tables. For *OpenSSL*, the discovered mean value is $1,303$ days old, whereas for *GnuPG*, the mean value is calculated to be more than $4$ years old ($1,653$ days).




(a) *OpenSSL* of outdated time-gap versions.

(b) *GnuPG* of outdated time-gap versions.

Figure 5.11: Letter-value plots of outdated time-gap versions.

### 5.5.2. Conclusions and Validity of results

This section presents the results and findings of discovered cryptographic libraries on firmware images using EBAT. Firmware images use broadly open-source cryptographic libraries in a percentage of over $75\%$, at least one cryptographic library is discovered embedded in the firmware image (except Libc (uClibc-ng [16] or glibc [5]) [48]). Additionally, the binaries that use the cryptographic libraries are more than $1$ million, over $13$ million of the total discovered ones. Despite the broad usage, outdated libraries are discovered in many firmware images, and even cryptographic libraries that have reached the *EoL* support even before the firmware image's publicly released date are found, with a percentage of nearly $50\%$ of the discovered cryptographic libraries being outdated and reaching their *EoL*.

The results and findings presented in this section are discovered only from publicly available firmware images with their release date as given publicly by the vendor's website in most cases. Thus, there is a low possibility of a few firmware images not having the correct public release date and our results being incorrect. Additionally, the extracted cryptographic library is the one that is found embedded in the firmware image as crawled and extracted from the vendor. The possibilities of updating these libraries as the device comes online or any other update mechanisms are not searched/covered. Thus, additional device update mechanisms may update the cryptographic libraries; however, the shipped firmware image remains outdated. The cryptographic version discovery mechanism has a low false positive rate as the version discovery is based on code analysis and not heuristics search string methods. Thus, the versions that are discovered are as precise as possible. Cryptographic library versions that are not discovered may produce a change in our results if they were discovered; however, less than $4\%$ of them still need to be discovered, and the change may be insignificant.

We mainly focused on two widely used cryptographic libraries discovered in our dataset of firmware images: the *OpenSSL* [14] and *GnuPG* [6]. *OpenSSL* and *GnuPG* are discovered at $77.85\%$ and $31.32\%$, respectively, over our successfully unpacked firmware images. Although *GnuPG* has a relatively high percentage of our firmware images, the usage of it that has been discovered in binaries is low, only $2.99\%$ of all the binaries that use a cryptographic library. On the other hand, *OpenSSL*, due to its wide usage, is used in over 45% of binaries. Comparing the total discovered cryptographic libraries for executables and libraries binaries, the percentages are split almost $60 - 40$, with a higher usage found in executable binaries. Furthermore, we investigate further the aforementioned cryptographic libraries with 2 scenarios that plot the histograms depicted in Figures 5.9a, 5.11a and 5.9b, 5.11b, for *OpenSSL* and *GnuPG* respectively. Unfortunately, we can observe that a large percentage of firmware images are deployed outdated, and also, a few of them have versions that have been outdated for consecutive years, which may lead to n-day attacks. Speculations about the lack of constant updating of the developer's tools and libraries and the usage of identical toolchains and OEM products raise concerns that this will eventually lead to outdated cryptographic libraries being spread across multiple firmware images and different products.

## 5.6. Common Vulnerabilities and Exposures (CVEs)

In this section, an analysis of EBAT's ability to find Common Vulnerabilities and Exposures (*CVEs*) that are listed in the CVE database for cryptographic libraries is presented. Although EBAT has the ability to find *CVEs* not only for cryptographic libraries but also for various types of libraries and executables such as busybox, zlib, libpng, libjpeg-turbo, libvorbis, et al., those results will not be presented as the discovery of their version depends only on CVE Binary Tool [51] as analyzed in Chapter 4.7 which may provide false positives and left for future work. On the other hand, cryptographic library CVEs also rely on an implemented module of EBAT that performs code analysis to discover the version and the results are considered more reliable. It should be noted that not all discovered *CVEs* will affect the firmware image immediately, as the discovery of a CVE is only an indication and not an immediate vulnerability on the firmware image but on the discovered library. Firmware images may patch the CVE or possibly not use this exact vulnerable code section; thus, further manual review needs to be done. These CVE indications are the first step to help the developer further secure their developed firmware image.

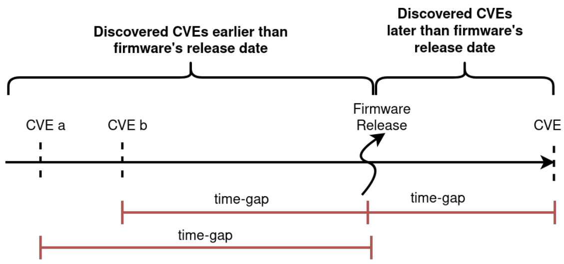


Figure 5.12: Scenario of discovered CVE.

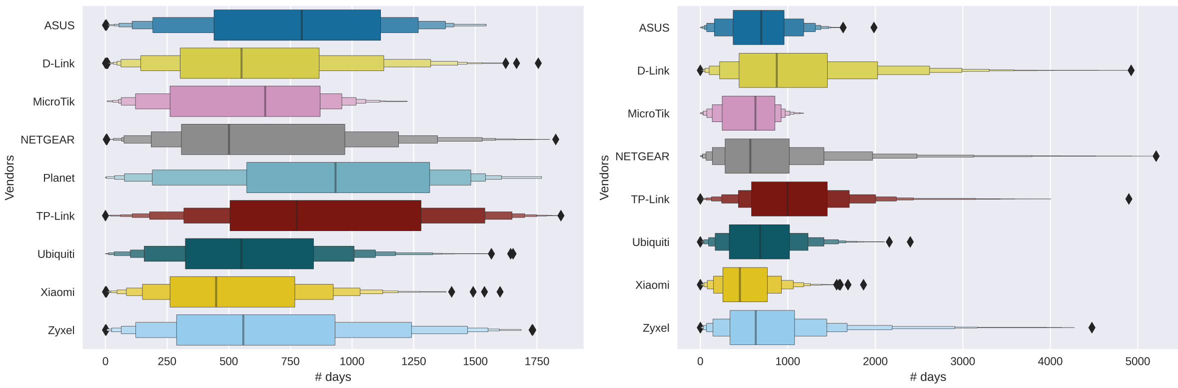
The CVEs that are discovered are separated into two categories: the ones that are known even before the publicly available firmware image release date and those that are released later than the publicly available firmware image release date. Figure 5.12 depicts the aforementioned two scenarios where CVE $a$ and CVE $b$ are presented even before the firmware image has been released, and CVE $c$ is discovered after the firmware's public release date. In our results, the time-gap will be measured in days. For each CVE along with the released date, the severity level is also saved as *Critical*, *High*, *Medium*, *Low*, harvested from the National Vulnerability Database (NVD) [74]. Table 5.9 presents the unique CVEs discovered for all firmware images earlier/later than a firmware's image release date for each cryptographic library. Those known CVEs may be found in more than one firmware image. Appendix Table A.37 presents the *Critical*, *High*, *Medium* and *Low* severity CVE for the cryptographic libraries examined in our work, by CVE number (for example, CVE-2020-12345) and severity, wherealso presents the number of firmware images a particular CVE is discovered.

| | Earlier | | | | Later | | | |
|---|---|---|---|---|---|---|---|---|
| | Critical | High | Medium | Low | Critical | High | Medium | Low |
| Library | # CVEs | # CVEs | # CVEs | # CVEs | # CVEs | # CVEs | # CVEs | # CVEs |
| GnuPG [6] (libgcrypt) | 0 | 2 | 7 | 3 | 0 | 2 | 6 | 3 |
| GnuTLS [7] (libgnutls) | 3 | 14 | 23 | 0 | 3 | 11 | 16 | 0 |
| KerberosV5 [93] (libk5crypto) | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| LibTomCrypt [10] (libtomcrypt) | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| mbedTLS/PolarSSL [11] (libmbedcrypto, libmbedtls, libpolarssl, libmbedx509) | 3 | 8 | 15 | 0 | 3 | 8 | 8 | 0 |
| Nettle [13] (libnettle) | 3 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| OpenSSL [14] (libcrypto, libssl) | 8 | 45 | 128 | 13 | 8 | 40 | 105 | 11 |
| WolfSSL [17] (libwolfssl, libcyassl) | 4 | 7 | 14 | 0 | 4 | 6 | 11 | 0 |
| **Total** | 21 | 78 | 189 | 16 | 18 | 67 | 148 | 14 |

Table 5.9: Overall **distinct CVEs** founds earlier/later than firmware images release dates per cryptographic library.

As we can observe from Table 5.9, 21 *Critical* and 78 *High* severity CVEs are found earlier than the release date of a firmware image, where 18 *Critical* and 67 *High* severity CVEs, are discovered later than the release date of a firmware image. Furthermore, according to Appendix Table A.37, more than 6,000 firmware images are possibly susceptible to at least one of those *Critical* CVEs, and more than 12,000 firmware images are possibly susceptible to at least one *High* severity CVEs, even before the firmware image goes publicly available. That is approximately 30% (*Critical*) and 56% (*High*) of the total unpacked firmware images. Appendix Table A.35 depicts the most popular *Critical* severity discovered CVEs, which are found to be in the OpenSSL cryptographic library, which are $CVE-2016-2177$, $CVE-2016-6303$, $CVE-2016-2182$, $CVE-2016-2108$, $CVE-2016-0705$ and $CVE-2016-0799$, that may affect more than 4,000 firmware images. That *Critical* CVEs affect the OpenSSL cryptographic library over multiple versions, such as $1.0.1a, 1.0.1b, 1.0.2a$, which can cause a denial of service (DoS) and/or even arbitrary code execution in some cases if the firmware image is susceptible to those known CVEs, making an n-day attack possible.

Figure 5.13a and 5.13b depict letter-value plots as calculated by Figure 5.12 that depicts the way how the time-gap in days of CVEs are calculated, for *Critical* and *High* severity CVEs, respectively. For every firmware image, the time-gap for CVEs earlier than the firmware's release date is calculated in days and plotted for each vendor separately. Appendix Tables A.11, A.12, A.13, A.14 depict the aforementioned calculated letter-value plots for all vendors in our dataset for *Critical*, *High*, *Medium* and *Low* severity, respectively.




(a) Critical severity CVEs time-gap in days earlier than the firmware release date.

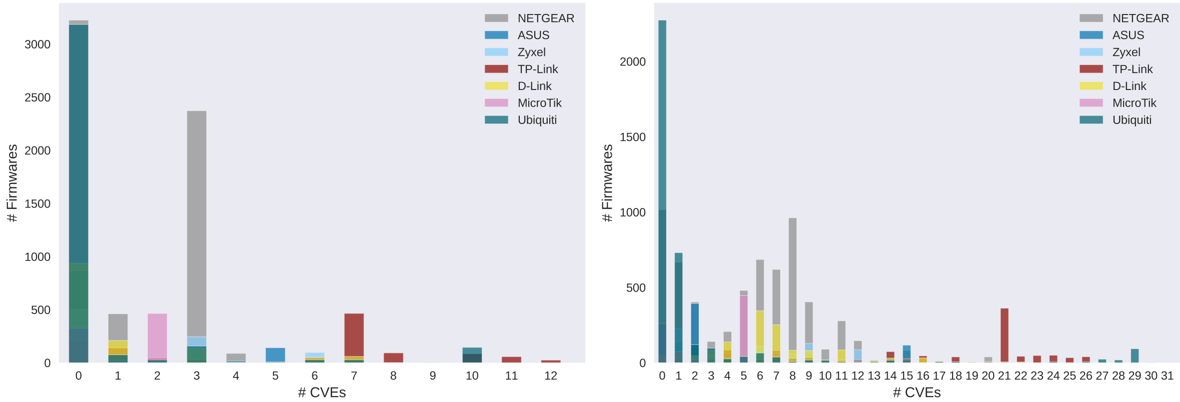
(b) High severity CVEs time-gap in days earlier than the firmware release date.

The mean values for Figures 5.13a and 5.13b, along with the mean values for all vendors, are presented in Appendix Tables A.33 and A.34, respectively. The median value in days extracted from vendors for *Critical* severity CVEs is calculated to be 590 days, which means that even more than one and a half years before even releasing the firmware image, at least one critical CVE exists, affecting one of the outdated cryptographic libraries. For *High* severity CVEs, the median value is longer and calculated to be more than 2 years (approximately 874 days). The critical severity CVEs discovered affect fewer firmware images than high to low-severity CVEs. It is crucial to recognize that, despite the severity of these CVEs, a firmware image may not be vulnerable if the specific vulnerable code part is not used.

Figures 5.14a and 5.14b plots histograms for *Critical* and *High* severity CVE. The plots represent the number of firmware images found to have 0 CVE, 1 CVE, 2 CVEs, etc., for vendors more dominant in our dataset. In total, from the histogram figures, 15,757 firmware images have 0 *Critical* CVE found (a percentage of 70%) earlier than the firmware image public release date, while the rest, meaning that 6,773 of the successfully unpacked firmware images in our dataset have at least one *Critical* severity CVE, in one of the installed cryptographic libraries. The same holds true for *High*, *Medium* and *Low* severity CVEs, although with even higher percentages than the *Critical* ones.

### 5.6.1. Conclusions and Validity of results

The results presented in this section are aggregated for every vendor's firmware images. A developer/researcher may run individually our implemented tool `EBAT` for a specific firmware image to identify if CVEs exist (not only on cryptographic libraries) in their examine firmware image where there is a pos-




(a) Histogram of critical severity CVEs earlier than the firmware release date.

(b) Histogram of high severity CVEs earlier than the firmware release date.

Figure 5.14: Letter-value plots and histograms for CVEs.

sibility of an existence of a CVE to lead to a vulnerability on the specific firmware image. We focused on CVEs only from open-source cryptographic libraries, affecting particular versions. The CVEs are categorized as *Low*, *Medium*, *High*, *Critical* and separated into two categories, the one that is discovered even before the firmware image has a publicly released date and the one after their release date. We mainly focused on *High* and *Critical* severity CVEs; however, all the severity results are presented thoroughly in Appendix Section A.7.

The previous section on version discovery of cryptographic libraries also correlates with the analysis presented in this section in CVEs for the discovered cryptographic libraries. As the versions are outdated, more CVEs are discovered even earlier than the publicly released date of a firmware image. Additionally, CVEs found later than the publicly released date of a firmware image are also presented, and an additional analysis needs to be done on those to check if they were used between firmware versions. However, this is left for future work. Unfortunately, a high percentage of *Critical* and *High* severity CVEs are discovered prior to the firmware image release date with approximately 30% of our successfully unpacked firmware images having at least one *Critical* CVE that may lead to an n-day attack if the particularly vulnerable code is in use.

All the discovered CVEs in this section rely on the discovered versions of cryptographic libraries that are embedded in the initial firmware image. Our results correlate with the previous Section 5.5, as the versions of the cryptographic libraries are analyzed. Code or physical device analysis was not performed for any specific CVE to validate the effectiveness of the discovered CVE to an actual n-day attack. A device may have the mechanism to update these libraries along the way. Additionally, some vendors may apply patches to known CVEs of the cryptographic libraries instead of updating them and/or the particular CVE vulnerable code may not be triggered (used) throughout the affected firmware image; however, the vulnerable code may exist. Thus, additional manual code audits and specific firmware analyses with device acquisition must be performed to verify the presented CVEs' validity.

## 5.7. Cryptographic Misuses

This section presents the results and findings of each cryptographic misuse rule discovered in the firmware image binaries. `EBAT's` static taint analysis module is used in order to detect cryptographic violations of the 18 cryptographic misuse rules as defined in Chapter 2.3, in which the static taint analysis implementation is presented in depth over multiple sections of Chapter 4 (4.9, 4.10, 4.11, 4.12 and 4.13). *Ghidra SRE* release version ***9.1.2 (02/2020)*** [70] along with `EBAT's` implemented modules and *Ghidra's* implemented headless scripts are used for producing the results that will follow. At the time of writing, newer Ghidra versions are being released, possibly providing better and more concrete results than our current results and findings. Newer versions of *Ghidra SRE* will be tested in future work as the implemented static taint analysis module is forward compatible.

Generally, for every binary of each successfully unpacked firmware image, static taint analysis is executed only if it is using one or more dynamically linkable cryptographic library/libraries, as presented in Chapter 4.6 Table 4.1. We created rules for our taint analysis, explained with more details in Chapter

4.11, where every rule presents the tainted function and tainted function arguments (if any) for each chosen examined function. These rules give us the necessary information for performing the start of static taint analysis and possibly discovering a violation. Additionally, vendors' cryptographic implemented wrappers are captured and followed to enhance our analysis further. The rules created for our taint analysis are tainting $733$ cryptographic functions with more than $1,500$ arguments that belong to $10$ cryptographic libraries well-defined Application Programming Interfaces (***APIs***). Those tainted arguments are then followed (taint propagation) until a taint sink is discovered or not. We are interested in discovering a violation (a misuse condition) of a cryptographic misuse rule as analyzed in Chapter 2.3. Overall, we examined $18$ common cryptographic misuse rules, which can be categorized by their cryptographic primitives as presented in Chapter 2.3. It is essential to mention that the analysis performed is static and not dynamic, and the following results will be based only on the executable binary's entry point, the call graph, the $\phi$ nodes, and any shared libraries. No dynamic or any other kind of intrusive analysis is executed on any publicly available devices to verify our claims, and further manual work must be addressed to verify any of the presented results. Bear in mind that our module is modular, where anyone can write their own rules, and the list of tainted rules is not by all means exhaustive. The results and findings will be aggregated over our entire dataset and presented for the rest of this section; case studies on real-world vulnerabilities will be examined and compared with `EBAT's` feasibility to discover them.

In order to limit the **false positives** of cryptographic misuses analysis, the following scenario is taken into account: If a function call is discovered implementing a cryptographic rule incorrectly, although it is never called from the entry point of any examined executable binary (i.e., *main*), the findings of those cryptographic violations will not be presented and discarded in this work. Listing pseudo-code 5.1 depicts the scenario mentioned earlier as an example. Listing pseudo-code 5.2 presents a simplified example of a cryptographic violation that is called from *main* (binary's entry point), and the findings of this scenario will be presented.

```c
// entry point
int main(int argc, char **argv){
  /* code */
  ...

  return 0;
}

// code either in the same binary or
 in a shared library
// violated rule S2 (encrypts more
than 1 block of data)
void AES_ECB_128_encrypt(...){
  /* code */
  ...
}
```

Listing 5.1: Function that **is not called** from a binary's entry point.

```c
// entry point
int main(int argc, char **argv){
  /* code */
  ...

  AES_ECB_128_encrypt(...);

  return 0;
}

// code either in the same binary or
 in a shared library
// violated rule S2 (encrypts more
than 1 block of data)
void AES_ECB_128_encrypt(...){
  /* code */
  ...
}
```

Listing 5.2: Function that **is called** from a binary's entry point.

In some cases, functions may be called from a binary's entry point, which can cause a cryptographic misuse. Those function calls are taken into account, although there is a possibility of false positives where the misuse function call may never be called as is. It may depend on other parameters as well as input parameters. On the other hand, a cryptographic violation is presented in the binary code, and the cryptographic misuse may occur at some point, depending on the execution path. For instance, Listing pseudo-code 5.3 depicts a scenario in which a cryptographic misuse may be called from the binary's entry point only if a particular path (in the listing example when $x = 0$) is triggered. In this scenario, taint analysis does **not marked** the sink node as $\phi$ **(phi)**, defined in Chapter 4.13, because the tainted argument does not get multiple results from various paths but is depended on an input argument that we cannot resolve using static code analysis. On the other hand, as an example, Listing pseudo-code 5.4 presents a cryptographic violation only when $m = EVP\_ECB()$ (path is triggered when

$x = 0$), where otherwise the other sink nodes are not violating any of the cryptographic misuse rules ($m = EVP\_CBC()$ or $m = EVP\_CTR()$). All the sink nodes and the violated one presented in the listing 5.4 are saved and **marked as $\phi$ (phi)**.

```
int main(int argc, char **argv){
  /* code */

  // read arbitary input
  x = readInput();

  if (x == 0){
    // violate rule S2 if called
    AES_ECB_128_encrypt(...);
  }
  else if (x == 1){
    /* code */

  }
  else{
    /* code */

  }

  return 0;
}

// code either in the same binary or in a shared library
// violated rule S2 (encrypts more than 1 block of data)
void AES_ECB_128_encrypt(...){
  /* code */

}
```

Listing 5.3: Function that may called from a binary's entry point.

```
int main(int argc, char **argv){
  /* code */
  // read arbitary input
  x = readInput()

  if (x == 0){
    // violated rule S2
    // ECB mode of operation
    m = EVP_ECB();
  }
  else if (x == 1){
    // no violation
    // CBC mode of operation
    m = EVP_CBC();
  }
  else if (x == 2){
    // no violation
    // CTR mode of operation
    m = EVP_CTR();
  } // may be others too

  AES_encrypt(m, ...);
  /* code */
  return 0;
}

// code either in the same binary or in a shared library
// violated rule S2 if called with "ECB" (encrypts more than 1 block of data)
void AES_encrypt(m, ...){
  /* code */
}
```

Listing 5.4: $\phi$ nodes as constant values, passing as an argument.

The aforementioned scenarios are not by all means exhaustive, as there are many combinations of them. Our tool tries to resolve all possible combinations and marks the $\phi$ nodes whenever we discover one. The results will be presented later in the rest of this section and separated into two categories: First, the ones that take into account that the discovered cryptographic misuse has an execution path from an entry point and there is a possibility of a $\phi$ node. Second, the ones that also have an execution path from an entry point and it is 'most likely' that it is not a $\phi$ node. Keep in mind that the analysis is not sound; thus, it is not certainly true that a $\phi$ node may exist, and our analysis missed it, as there may be an execution path that is not covered in our analysis.

### 5.7.1. Overall results for Cryptographic Misuses

Overall, the taint analysis for detecting cryptographic misuses is executed in over $1,4$ **million binaries** that belong to $22,548$ successfully unpacked firmware images (including $424$ partially unpacked firmware image). Appendix Table A.6 presents the overall statistics of analyzed binaries for each vendor. In total, only $4.37\%$, approximately $1.4$ million binaries are analyzed for cryptographic misuses as they are the ones that are using one or more cryptographic dynamically link libraries. The covered cryptographic libraries are presented in the previous chapter Table 4.1. Executables are $60\%$ of the total analyzed binaries; the rest are libraries.

Table 5.10 depicts the percentages of discovered cryptographic primitives found from binaries entry point over all successfully unpacked firmware images in our dataset represented in the 2nd column. The percentage in parenthesis is presented over the $22,548$ successfully unpacked firmware images.

In the 3rd column, we present the number of binaries found the specific cryptographic primitive from entry over all examined executable binaries, where the percentage in parenthesis is calculated over 861,946 (see A.6) executable binaries. 'AES-n/a' is the usage of AES in which the key length is not discovered. We can generally observe for Symmetric Key Encryption, that the usage of AES [75] is prevalent; however, non-secure ciphers such as *DES* and stream ciphers such as *RC2* are also present in firmware images. Additionally, Key Derivation Functions uses *DES (KDF - DES)* in a percentage of over 25% for all the examined firmware images. Moreover, a high usage of *MD-5* is observed, a bit over the *SHA-1* hash algorithm. Lastly, *HMAC*, *RSA*, *Elliptic Curves (EC)* and *X.509* cryptographic functions are discovered to be in use from one out of four firmware images.

Table 5.11 depicts the aggregated discovered cryptographic rules violations for every firmware image successfully unpacked in our dataset. For each cryptographic primitive, the percentage of ***'no violation'*** is calculated, which counts the number of firmware images in which not a single violation is discovered. Furthermore, a ***'total no violation'*** is calculated, which shows the number of successfully unpacked firmware images that not a single violation is discovered for any of the rules, excepting rules ***R1*** and ***R2*** **(analyzed later)** and a total no violation count for any of the rules. Additionally, the results are presented for two categories, ***'entry and possible $\phi$' nodes*** and ***'entry and not discovered $\phi$' nodes***. *'Entry and possible $\phi$' nodes* are the cryptographic misuses discovered from a binary's entry point, and one or more $\phi$ nodes exist. *'Entry and not discovered $\phi$' nodes* are the cryptographic misuses discovered from a binary's entry point, and EBAT does not discover any $\phi$ node associated with this cryptographic misuse. The following subsections will examine individual results for each rule separately.

| Cryptographic Primitives | # Firmwares discovered | # Binaries discovered |
|---|---|---|
| Symmetric Key Cryptography | | |
| AES-n/a | 8,753 (38.82%) | 14,791 (1.72%) |
| AES-128 | 9,988 (44.30%) | 20,331 (2.36%) |
| AES-192 | 904 (4.01%) | 1,129 (0.13%) |
| AES-256 | 3,236 (14.35%) | 3,989 (0.46%) |
| BLOWFISH | 1,321 (5.86%) | 1,408 (0.16%) |
| CAMELLIA | 450 (2.00%) | 849 (0.10%) |
| CAST | 9 (0.04%) | 10 (0.00%) |
| CAST5 | 20 (0.09%) | 21 (0.00%) |
| DES | 6,370 (28.25%) | 11,279 (1.31%) |
| TDES2 | 8 (0.04%) | 8 (0.00%) |
| TDES3 | 2,245 (9.96%) | 2,266 (0.26%) |
| GOST | 6 (0.03%) | 6 (0.00%) |
| IDEA | 13 (0.06%) | 14 (0.00%) |
| RC2 | 9,71 (4.31%) | 1,301 (0.15%) |
| RC4 | 6,189 (27.45%) | 22,629 (2.63%) |
| Authenticated encryption/decryption and AEAD | | |
| AES-CMAC | 222 (0.98%) | 488 (0.06%) |
| AES-GCM | 172 (0.76%) | 190 (0.02%) |
| CHACHA20-POLY1305 | 26 (0.12%) | 26 (0.00%) |
| Message Authentication Codes (MACs) | | |
| HMAC | 16,562 (73.45%) | 25,782 (2.99%) |
| Key Derivation Functions (KDFs) | | |
| BCRYPT | 25 (0.11%) | 25 (0.00%) |
| KDF DES | 6,192 (27.46%) | 11,829 (1.37%) |
| KDF | 445 (1.97%) | 508 (0.06%) |
| PBKDF2 | 19 (0.08%) | 19 (0.00%) |
| Public Key Cryptography | | |
| EC | 5,671 (25.15%) | 10,129 (1.18%) |
| RSA | 16,517 (73.25%) | 51,441 (5.97%) |
| X.509 | 5,912 (26.22%) | 7,183 (0.83%) |
| Cryptographic one-way hash functions | | |
| MD4 | 3,787 (16.80%) | 6,276 (0.73%) |
| MD5 | 13,337 (59.15%) | 77,529 (8.99%) |
| SHA | 11 (0.05%) | 11 (0.00%) |
| SHA1 | 10,928 (48.47%) | 48,126 (5.58%) |
| SHA224 | 301 (1.33%) | 341 (0.04%) |
| SHA256 | 4,025 (17.85%) | 7,366 (0.85%) |
| SHA384 | 719 (3.19%) | 884 (0.10%) |
| SHA512 | 869 (3.85%) | 1,790 (0.21%) |
| BLAKE2B | 2 (0.01%) | 2 (0.00%) |
| RIPEMD160 | 137 (0.61%) | 137 (0.02%) |

Table 5.10: Discovered cryptographic primitives over all firmware's binaries (**found a call from entry**).

Appendix Tables A.38, A.39, A.40, A.41, A.42 and A.43 present the cryptographic misuses that are discovered from an entry node, and there is a possibility of being a $\phi$ node for every vendor in our dataset separately, for Symmetric Key Cryptography, Public Key Cryptography, Pseudo Random Number Generators (PRNGs), Key Derivation Functions (KDFs) and Password Based Encryption (PBE), Message Authentication Codes (MACs) and Authenticated encryption/decryption and AEAD respectively. Finally, Appendix Tables A.44, A.45, A.46, A.47, A.48 and A.49, presents the cryptographic misuses that are discovered from an entry node and not a single $\phi$ node is discovered, for every vendor in our dataset separately, for Symmetric Key Cryptography, Public Key Cryptography, Pseudo Random Number Generators (PRNGs), Key Derivation Functions (KDFs) and Password Based Encryption (PBE), Message Authentication Codes (MACs) and Authenticated encryption/decryption and AEAD respectively. A separate section will follow for each one, analyzing the results and providing more context.

| Rule # | Short Description | entry and possible $\phi$ | | entry and not discovered $\phi$ | |
|---|---|---|---|---|---|
| | | # Firm. violate | % | # Firm. violate | % |
| | Symmetric Key Cryptography | | | | |
| ***S1*** | Constant Encryption/Decryption Keys | 569 | 2.52% | 569 | 2.52% |
| ***S2*** | Usage of ECB mode of operation | 3,794 | 16.39% | 3,794 | 16.39% |
| ***S3*** | Constant IV for various modes of operation | 21 | 0.09% | 21 | 0.09% |
| ***S4*** | Usage of 'weak' ciphers for encryption | 2,830 | 12.55% | 2,830 | 12.55% |
| No violation of ***S1***, ***S2***, ***S3***, ***S4*** | | 17,889 | 79.34% | 17,889 | 79.34% |
| | Public Key Cryptography | | | | |
| ***P1*** | Usage of insecure RSA encryption padding schemes | 3,850 | 17.07% | 3,850 | 17.07% |
| ***P2*** | DSA usage of 'weak' digest function | 0 | 0.00% | 0 | 0.00% |
| ***P3*** | X.509 certificate usage of 'weak' digest function | 1,624 | 7.20% | 1,604 | 7.11% |
| No violation of ***P1***, ***P2***, ***P3*** | | 17,951 | 79.61% | 17,971 | 79.70% |
| | Pseudo Random Number Generators (*PRNGs*) | | | | |
| ***R1*** | PRNG static seed | 3,333 | 14.78% | 2,491 | 11.05% |
| ***R2*** | Low entropy sources for seeds | 11,630 | 51.58% | 11,601 | 51.45% |
| No violation of ***R1***, ***R2*** | | 8,737 | 38.75% | 9,021 | 40.01% |
| | Key Derivation Functions (*KDFs*) and Password Based Encryption (*PBE*) | | | | |
| ***K1*** | Constant Passwords on a KDF/PBE | 898 | 3.98% | 316 | 1.40% |
| ***K2*** | Constant salt or no salts on a KDF/PBE | 3,173 | 14.07% | 1,901 | 8.43% |
| ***K3*** | 'Weak' number of iteration on a KDF/PBE | 445 | 1.97% | 441 | 1.96% |
| ***K4*** | 'Weak' underlying hash function on a KDF/PBE | 8,068 | 35.78% | 6,662 | 29.55% |
| No violation of ***K1***, ***K2***, ***K3***, ***K4*** | | 13,632 | 60.46% | 15,513 | 68.80% |
| | Message Authentication Codes (*MACs*) | | | | |
| ***M1*** | Constant Encryption/Decryption Keys on a MAC | 332 | 1.47% | 275 | 1.22% |
| ***M2*** | 'Weak' underlying hash function on a MAC | 1,800 | 7.98% | 1,646 | 7.30% |
| ***M3*** | Non-secure key length on a MAC function | 80 | 0.35% | 79 | 0.35% |
| No violation of ***M1***, ***M2***, ***M3*** | | 20,421 | 90.57% | 20,615 | 91.43% |
| | Authenticated encryption/decryption and *AEAD* | | | | |
| ***A1*** | Constant Encryption/Decryption Keys on AE/AEAD | 0 | 0.00% | 0 | 0.00% |
| ***A2*** | Constant IV on AE/AEAD | 0 | 0.00% | 0 | 0.00% |
| No violation of ***A1***, ***A2*** | | 22,548 | 100% | 22,548 | 100% |
| | Total | | | | |
| No violation of any of the rules above (except ***R1*** and ***R2***) | | 10,885 | 48.27% | 12,040 | 53.40% |
| | Total | | | | |
| No violation of any of the rules above | | 6,639 | 29.44% | 7,469 | 33.12% |

Table 5.11: Overall discovered Cryptographic Rules Violations (**Cryptographic Misuses**) in our entire firmware images dataset.

#### *5.7.1.1 Cryptographic Misuses in Symmetric Key Cryptography rules*

| Type | # functions | # arguments |
|---|---|---|
| **S1** | 88 | - |
| **S2** | 72 | - |
| **S3** | 65 | - |
| **S4** | 74 | - |
| Other | 27 | - |
| **S1, S2, S3, S4** and other | 326 | 880 |

Table 5.12: Number of tainted functions and arguments for Symmetric Key Cryptography rules.

Overall, for Symmetric Key Cryptography, we discovered a total of 79.34% of all the successfully unpacked firmware images not to violate a single Symmetric Key Cryptography rule, where the rest (20% approximately) are violating at least one of the following four rules: **S1, S2, S3, S4** as presented in Table 5.11. Table 5.12 depicts the total tainted functions for symmetric key cryptography for rules **S1, S2, S3, S4** and the total ones. Total tainted arguments are also reported. Other functions present the tainted rules that helped us detect a rule or provide additional context to the rule we examined, such as a key/iv *length* to perform a symmetric key encryption/decryption. Only the total function arguments for all the examined rules and other functions are presented since there is no need to report the arguments separately for specific rules as they are dependent on other arguments for detecting a violation but also for detecting any additional context.

In more depth for constant encryption/decryption keys (rule **S1**), the most popular tainted functions for all of our examined binaries are found to belong in *OpenSSL* cryptographic library, which is the one most commonly used. The function that is tainted the most with our given rules throughout our results is `EVP_CipherInit`, which, most of the time, the discovered key is found to be `NULL` as the symmetric key is applied in other functions throughout its context. The most violated functions, either directly or through a wrapper, are discovered to be `AES_set_encrypt_key` and `AES_set_decrypt_key`. A few discovered constant keys are found to be weak, such as `'root123'` and `'1234567890abcdef'`, where we have managed to discover more symmetric keys that we cannot disclose in this work as there are not publicly known and this may compromise the security of a device. All the keys reported by our tool are in base64, as some symmetric keys are not ASCII printable. Both for *'entry and possible $\phi$'* and 'entry and not discovered $\phi$' the results are identical, 569 firmware images discovered to be violating rule **S1**, only a small percentage of examined firmware images, 2.52%. The results of both cases are identical, meaning we do not discover a $\phi$ node that the particular discovered key can change for the particular function call. Bear in mind that other functions may exist in the binary setting a symmetric key securely, and the cryptographic misuse function may never be called. However, the discovered execution path can be called from the binary's entry point.

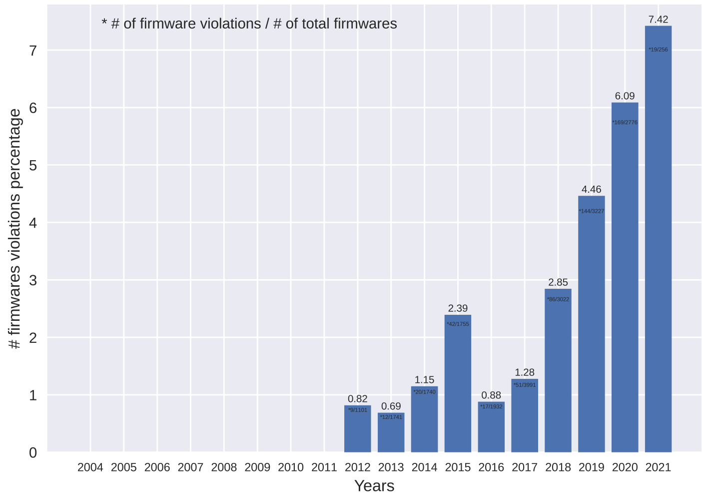


Figure 5.15: Discovered Cryptographic misuses over the years for rule **S1**.

Figure 5.15 depicts a percentage over the years, which is the number of firmware images divided by the total successfully unpacked firmware images that a cryptographic misuse for rule **S1** is discovered (violated) by each year in our dataset. Appendix table A.2 depicts the number of successfully unpacked firmware images aggregated by year. The $y$ axes represent the percentage of firmware images with at least one violation of rule **S1** over the total successfully unpacked firmware images, and the $x$ axes represent each year. Year 2021 is incomplete and can be considered an outlier as our dataset ends in the early second quarter of 2021. We can observe from the figure that discovering the symmetric keys embedded in firmware images rose over the years (in percentage terms), especially in the year 2020 with a total of 6.09%, 169/2776. One reason for the increase may lie in our dataset, as we cannot obtain as many firmware images as they have not been publicly disclosed throughout the years by the vendor or may have been removed to replace new ones. Nevertheless, the results are presented as is and are specific to the successfully unpacked firmware images that our tool analyzed.

Appendix Table A.50 presents the name of the violated binaries that a violation is discovered along with the number of firmware images that the specific binary name is discovered, ordered by each vendor. The most used executable binaries names that a violation is discovered for rule **S1** are: 'Netgear_ddns', 'imgdecrypt', 'smm', 'firebase', 'tdpServer', 'mainfunction.cgi', 'goahead', 'oneTimeCall' and 'securitypage', which are discovered in 144, 134, 96, 96, 62, 44, 30, 18 firmware images, respectively. The exact context of those binaries needs further analysis as one can speculate from the name of these binaries to be mainly in the network communication context. In the case study 5.7.2.1 that will presented in a later section of this chapter, the cryptographic misuse of rule **S1** will be explained on the specific violation discovered on the binary 'imgdecrypt' for D-Link products, which is a binary responsible for decrypting the firmware image.

The usage of *ECB* mode of operation, violating rule **S2**, may produce false positives, despite our efforts due to the complexity of calculating the actual usage of discovered *ECB* functions, meaning that the *ECB* discover function calls may be discovered from a binary's entry point; however, the execution path may never trigger if it depends on any other parameter, as shown in Listing 5.3. Additionally, limiting the usage of the functions that encrypt only a single block of data is a challenging task which is left for future work. Although there is a possibility of encrypting one block of data without violating rule **S2**, the results are presented as is. Both for *'entry and possible $\phi$'* and 'entry and not discovered $\phi$', the results are discovered to be identical for rule **S2**. A possible reason is that the discovered function is standalone, meaning if it is found to be used, no $\phi$ path of this function can be discovered. Additionally, for functions that declare the mode of operation in an argument, our results show that the specific path of this argument does not lead to another variant of the mode of operation. It is important to note that this does not mean that the only mode of operation is *ECB* as there is a possibility of another mode of operation with different function calls that will lead to other paths, and eventually the discovered *ECB* mode will never use. The scenarios are endless; thus, multiple stages of analysis must be done in every specific firmware image to verify any possible violations.

The binaries discovered for violating **S2** strengthen the previous argument; the complete list can be found in appendix table A.50. The most discovered ones are: 'readyNASVault', 'afppasswd', 'wpa_supplicant', 'hostapd', 'cfg_client' and 'cfg_server' discovered a violation at least one time in 1038, 881, 754, 611, 367 and 357 firmware images respectively. The 'afppasswd' binary is responsible for allowing the maintenance of afppasswd files created by netatalk for use by the 'uams_randnum.so' library; thus, probably in the code arguments are configuration for *ECB* mode of operation, which is unclear if the firmware images are using it or not. Binary 'readyNASVault' is a proprietary binary from NETGEAR, which is unclear about the usage of *ECB* mode of operation where multiple stages of analysis needs to be done to confirm the usage or not. Other interesting binaries are also found to violate rule **S2**; some of them are 'img_backup', 'img_restore', 'synoappexport' and 'mariabackup', are proprietary firmware image binaries, which the first three are belonging to Synology, and the last one to TP-Link vendor.

Figure 5.16 depicts the percentage of firmware images that a cryptographic misuse for rule **S2** is discovered (violated) over the total successfully unpacked firmware images presented for multiple years in our dataset. We can observe that *ECB* mode of operation remains in many binaries in our dataset in the last cover years. The peak of 43.42% is observed in year 2013, and we can also observe a decline over the following years, although the usage of *ECB* mode of operation is discovered to be high with the last cover year in 2020 to be 24.14%. Year 2021 is not complete and is considered an outlier.

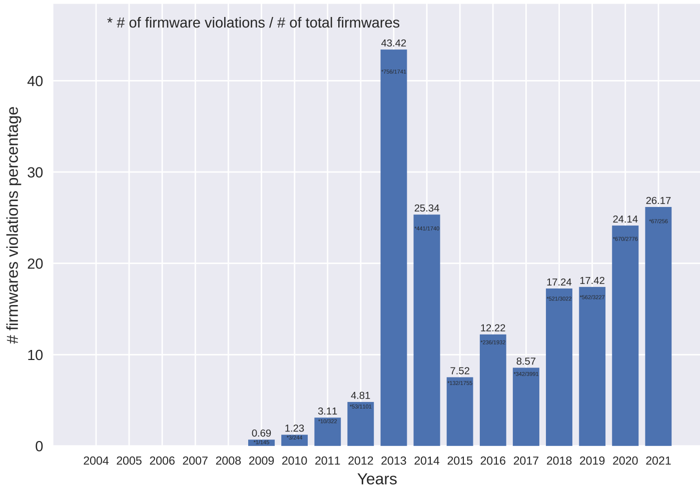



Figure 5.16: Discovered Cryptographic misuses over the years for rule **S2**.

Some of the most discovered functions that violate rule **S2** are found to be `EVP_des_ede3_ecb`, `EVP_aes_256_ecb`, `EVP_EncryptInit_ex` (with an *ECB* algorithm provided by the OpenSSL context), and `DES_ecb_encrypt`. Overall, 16.39% of the examined firmware images used at least one function in one of their binaries where there is a possibility of performing symmetric key encryption/decryption using the mode of operation as *ECB*.

Regarding constant IV (rule **S3**), the functions that are discovered for the 21 violations in firmware images are `EVP_DecryptInit` and `EVP_EncryptInit_ex` again from *OpenSSL* cryptographic library, with only 2 unique discovered fixed IVs. The first is found to be `'abcdefghijklmnop'` while the other is `'9kJmSY2bWumviYIM'`. These two fixed IVs are used in 21 firmware images, and no other $\phi$ path is discovered in our analysis. Thus, the results for both cases are identical. Furthermore, many discovered fixed/constant IVs are removed through manual analysis as we marked that as a false positive despite the tool reporting it as a violation. Two constant IVs, `'CJalbert'` and `'LWallace'`, were discovered multiple times in libraries called 'uams_dhx.so' and 'uams_dhx_pam.so'. However, our analysis did not find a single call from a binary's entry point; thus, our result will not reflect this.

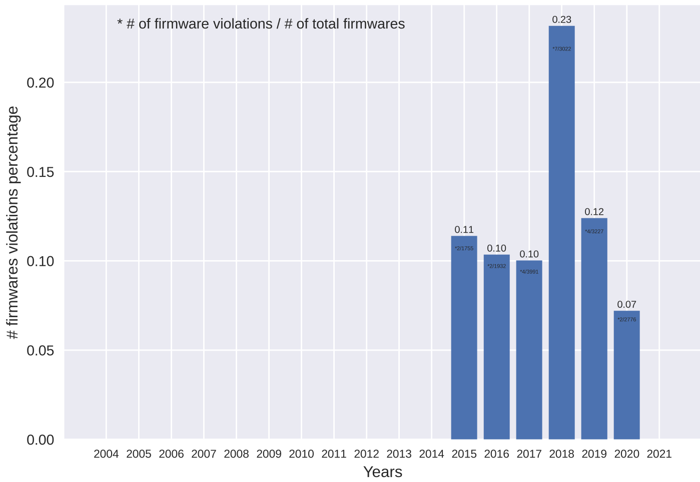



Figure 5.17: Discovered Cryptographic misuses over the years for rule **S3**.

Figure 5.17 depicts the percentage of firmware images that a cryptographic misuse for rule **S3** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images. The violation of using a constant IV was discovered for specific firmware images in a NETGEAR product and remained there throughout the years of consecutive firmware updates. The same holds for a D-Link product, which is also discovered individually in a few other D-Link firmware images. Lastly, the 21 violations of rule **S3** were discovered in 2 binaries ‘NetReadyAgent’ and ‘protest’, in 12 and 9 firmware images, respectively, where the first binary belongs to NETGEAR and the second to D-Link vendor.

At last, 2,830 (12.55%) firmware images are discovered to be violating rule **S4** for discovering the use of weak ciphers. In these results, only weak ciphers using symmetric key encryption are reported, and any decryption is discarded (assuming the possibility of usage on legacy devices). Thus, the reported results are filtered only for encryption. Repetitively, there is a chance that the execution path may never be executed, although references to those functions from the binary’s entry point are present. Our results are identical for *‘entry and possible $\phi$’* and ‘entry and not discovered $\phi$’ are discovered to be identical for rule **S4**. A possible reason is that the discovered function is standalone, meaning no $\phi$ path of this function can be discovered. Additionally, for functions that declare the symmetric cipher in an argument, our results show that the specific path of this argument does not lead to another variant of a symmetric cipher. It is important to note that this does not mean that the only cipher in use is a weak cipher; this is not true, and our tool also reports ciphers that are not weak and maybe are the ones that are in use. Thus, there is a possibility that another symmetric cipher with different function calls that are not weak will lead to other paths. Eventually, the discovered weak one will never be used. The scenarios are endless; thus, multiple stages of analysis must be done in every specific firmware image to verify any possible violations.

The binaries discovered for violating rule **S4** strengthen the previous argument, where the complete list is provided in appendix table A.50. The most discovered ones are: ‘wpa_supplicant’, ‘afppasswd’, ‘hostapd’, ‘wpad’, ‘snmpd’ and ‘fbwifi’ discovered a violation at least one time in 1073, 825, 641, 297, 286 and 208 firmware images respectively. The ‘wpa_supplicant’ is an executable binary responsible for the wireless connection of clients. It is unclear whether the firmware images use weak ciphers, as the wireless connection can be configured with strong ciphers. Thus, multiple stages of analysis of the specific firmware images needs to be performed. Furthermore, proprietary binaries are also discovered to violate rule **S4**. Some are ‘funjsq_cli’ and ‘upAgent’, belonging to NETGEAR.

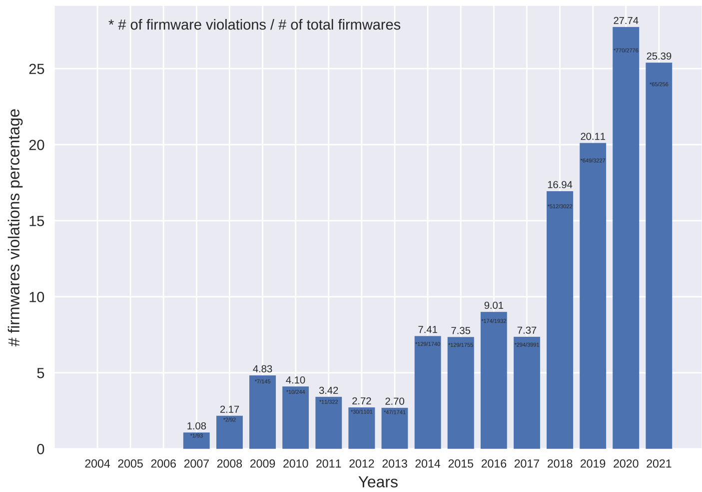


Figure 5.18: Discovered Cryptographic misuses over the years for rule **S4**.

Figure 5.18 depicts the percentage of firmware images that a cryptographic misuse for rule **S4** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images. We can observe that the usage of weak ciphers for symmetric key encryption is still found in many binaries in the last cover years of our dataset, with a peak in the year 2020. The most discovered functions for using a weak cipher and violating rule **S4** are found to be `EVP_EncryptInit_ex` and `EVP_CipherInit_ex` with context found to be a weak cipher encryption call of either *DES*, *RC2*, *RC4* or *Blowfish*. Additionally, multiple function calls from `DES_ncbc_encrypt` and `DES_ecb_encrypt` that are using the *DES* encryption algorithm are also discovered. *DES* symmetric encryption is the prevailing weak cipher discovered in our results, following in the specific order is *RC4*, *Blowfish*, *RC2* and *TDES2*.

#### 5.7.1.2 Cryptographic Misuses in Public Key Cryptography rules

Table 5.13 depicts the total tainted functions for Public Key Cryptography rules **P1, P2, P3** and the total ones. Total tainted arguments are also reported. Other functions present the tainted rules that helped us detect a rule or provide additional context to the rule we examined, such as a digest function used for rule **P2** and **P3**. Only the total function arguments for all the examined rules and other functions are presented since there is no need to report the arguments separately for specific rules as they are dependent on other arguments for detecting a violation but also for detecting any additional context. Overall, 17,951 firmware images, a total of 79.61% of the successfully unpacked firmware images, no violation for ‘entry and possible $\phi$ node’ case is discovered for the any of the Public Key Cryptography rules **P1**, **P2** and **P3**. For the ‘entry and not discovered $\phi$ node’ case, a total of $17,971$ $(79.70\%)$ firmware images are discovered not to violate any of the rules **P1**, **P2** and **P3**.

Specifically, for Public Key Cryptography rule **P1**, the usage of insecure *RSA* encryption padding schemes, we discovered $3,850$ firmware images $(17.07\%)$ of the successfully unpacked firmware images to use at least one insecure padding in their firmware images. The results are identical for both for *‘entry and possible $\phi$’* and ‘entry and not discovered $\phi$’ nodes. Table 5.14 represents the overall discovered *RSA* padding schemes that violate rule **P1** and the OAEP padding that is not violating the rule **P1**, in every binary for all successfully unpacked firmware images for all vendors. The 2nd column represents the number of references in all binaries regarding whether a violation is discovered. The 3rd column represents the number of firmware images found using the specific padding that resulted in a violation, and the 4th column is the number of firmware images that use the specific padding with no OAEP padding discovered on any binary in the specific firmware image. It should be noted that a firmware image can use multiple padding schemes, and the references that are not found from the entry may lead to a possible dead code.

| Type | # functions | # arguments |
|---|---|---|
| **P1** | 18 | - |
| **P2** | 11 | - |
| **P3** | 16 | - |
| Other | 33 | - |
| **P1, P2, P3** and other | 78 | 103 |

Table 5.13: Number of tainted functions and arguments for Public Key Cryptography rules.

The padding scheme *PKCS #1 v1.5* is the most dominant one with over $46$ thousand references, where $3,828$ firmware images result in a violation and from those, only $548$ firmware images are discovered to not use any OAEP padding anywhere else in their binaries. Furthermore, for *Optimal Asymmetric Encryption Padding (OAEP)* scheme is discovered in $4,446$ firmware images. No padding scheme is also discovered in RSA encryption (also called “textbook RSA”) in firmware images that result in a total of $22$ firmware images, and $7$ of those do not use any OAEP padding in all of their binaries. Unfortunately, $4$ firmware images belonging to NETGEAR and Totolink vendor are discovered not to use either *PKCS #1 v1.5* padding, and the only padding that is discovered by our tool is “textbook RSA”.

| Padding type | # references | # firmwares | # firmwares (not OAEP) |
|---|---|---|---|
| OAEP padding | 8,897 | 4,446 | - |
| PKCS #1 v1.5 padding | 46,402 | 3,828 | 548 |
| No padding | 955 | 22 | 7 |
| SSLv23 padding | 14 | 0 | 0 |

Table 5.14: Discovered `RSA padding` schemes for all firmware images.

Figure 5.19 depicts the percentage of firmware images that a cryptographic misuse for rule **P1** is discovered (violated) for multiple years in our dataset. The percentage is calculated over the violated firmware images divided by each year’s total successfully unpacked firmware images separately. We can observe that the firmware images that use insecure padding have significantly reduced over the years, with the peak year being in $2013$ of approximately $47\%$ and significantly dropping in the following years to reach $8\%$ in the year $2020$. The year $2021$ is not complete and can be considered an outlier. The top $7$ discovered binaries for violating rule **P1** are: ‘fvdropbox’, ‘avdu’, ‘fvamazon’, ‘readynasd’, ‘cfg_client’, ‘etm’ and ‘synolicense_uninstall’ discovered a violation at least one time in $2244$, $2014$, $1232$, $928$, $335$, $95$ and $52$ firmware images respectively. The first $4$ binaries belong to the NETGEAR vendor, while the other $3$ belong to ASUS, Xiaomi and Synology, respectively. It is unclear which one is proprietary binary, and we cannot examine their usage just by their name; multiple stages of analysis needs to be done. The complete list of violated executable binaries names discovered are presented in Appendix Table A.51 for the case of ‘entry and possible $\phi$’ and in Appendix Table A.55 for the case of ‘entry and not discovered $\phi$’. The most tainted functions discovered that vi-

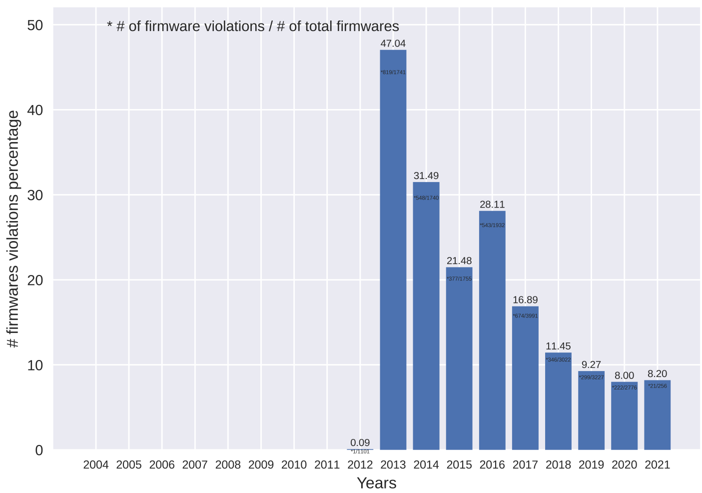



Figure 5.19: Discovered Cryptographic misuses over the years for rule **P1**.

olate rule **P1**, in our findings are from the *OpenSSL* cryptographic library where the functions are the following: `RSA_public_decrypt`, `RSA_private_decrypt` and `RSA_public_encrypt` that found to use the PKCS #1 v1.5 padding. The `RSA_public_encrypt`, `RSA_public_decrypt`, `RSA_private_decrypt` and `wc_RsaPrivateDecrypt` functions are the ones that found to use the no padding scheme.

Rule **P2** is for Digital Signatures signing/verifying with a "weak" underlying digest function. `EBAT` analysis found no violation in the examined firmware images. From Table 5.13, we can observe that only 11 tainted functions are used for rule **P2**, which may be one reason that no violation of this rule is discovered. The specific functions are only from the OpenSSL cryptographic library. This rule can be expanded to include more functions and function arguments for more cryptographic libraries as a future work.

Rule **P3** examined the X.509 certificate signing/verifying methods with a "weak" digest function. In total, 1,624 firmware images were discovered to violate this rule, a percentage of 7.20% over all the examined successfully unpacked firmware images. A slight difference is observed in the two examined cases. Thus, the following analysis is on the second case when an entry and possible $\phi$ node violations are discovered. Figure 5.20 depicts the percentage of firmware images that a cryptographic misuse for rule **P3** is discovered (violated) over multiple years in our dataset. The percentage is calculated over the violated firmware images divided by each year's total successfully unpacked firmware images separately. We can observe that the majority of firmware images that use an insecure cryptographic hash function were discovered in years 2013 and 2014, where the percentage dropped and remained nearly steady in the following years. Appendix tables A.51 and A.55 present the executable binaries names in the case of 'entry and possible $\phi$' and in the case of 'entry and not discovered $\phi$' respectively. The top 7 binaries discovered for violating **P3** are: 'ntfsdecrypt', 'certgen', 'monit', 'x509SelfSign', 'lftp', 'mpop' and 'httpd' discovered a violation at least one time in 860, 222, 136, 102, 88, 85 and 47 firmware images respectively. The 'ntfsdecrypt' binary decrypts a file from an unmounted device and prints the decrypted data on the standard output. It is found to have the option to read an X.509 certificate using an insecure digest. However, this is not a violation as it is an option, and the analysis cannot verify any input options. Thus, multiple stages of analysis needs to be done on the specific firmware image to determine the usage of this binary. This binary was discovered in NETGEAR and Western-Digital vendors. The 'certgen' binary generates a self-signed certificate tool with the option for an insecure digest. Similarly, with the 'ntfsdecrypt' binary, the usage is unknown, and we cannot directly declare it as a violation; thus, multiple stages of analysis is needed. This binary is found only in the NETGEAR vendor.

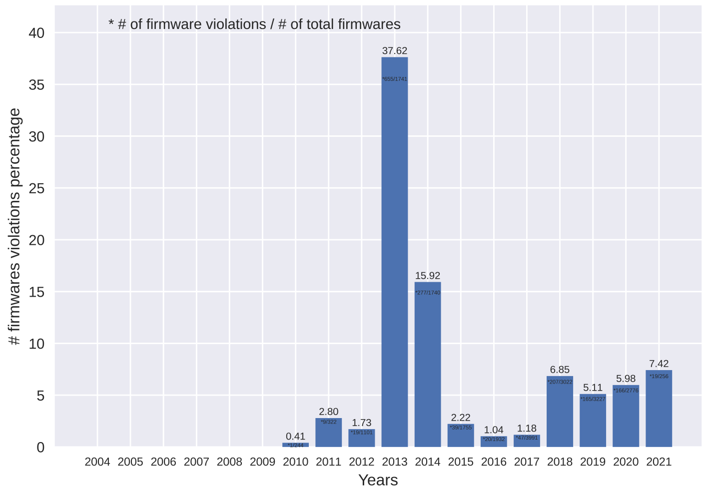



Figure 5.20: Discovered Cryptographic misuses over the years for rule **P3**.

In more depth, `SHA-1` and `MD-5` are the underlying cryptographic hash functions discovered for signing/verifying X.509 certificates for violating rule **P3**. The most violated tainted functions for this rule were discovered from the *GnuTLS* cryptographic library to be `gnutls_x509_crt_get_fingerprint` that will calculate and copy the certificate's fingerprint using a digest algorithm. For `MD-5`, the number of references was less than `SHA-1` for all violated binaries. Specifically, 372 firmware images were discovered to violate rule **P3** using the `MD-5` digest algorithm without any `SHA-1` digest. In contrast, 1145 firmware images were discovered to violate rule **P3** with `SHA1` without any use of `MD-5` digest algorithm. The remaining ones are found to violate this rule using both algorithms.

#### 5.7.1.3 *Cryptographic Misuses in Pseudo Random Number Generators rules*

The complexity of identifying the context usage of Pseudo Random Number Generators (PRNGs) for rules **R1** and **R2** makes the detection of potential violations more challenging. Cryptographic PRNGs are critical in cryptographic algorithms, key generation, and other security-sensitive tasks. However, distinguishing their use in such contexts from non-security applications demands multiple stages of analysis of specific functions when a possible violation is discovered in the firmware image. Additionally, there is a possibility of re-initialization of the seeds of PRNGs to a secure one, which makes the whole process harder. Thus, it may produce false positives in our results, as we do not know the specific context of using these functions and re-initialization of those functions in a different code path is also possible. That is why we decided to give results on the total no violations count from Table 5.11 that with rule **R1** and **R2** to be excluded. Additionally, we examined case studies for PRNGs in section 5.7.2.3 of this chapter, which resulted in a violation of rule **R2**. Overall, more than $8$ thousand firmware images are found not to violate any of the rules **R1** and **R2** resulting in a percentage of over $38\%$ for 'entry and possible $\phi$' and a slightly higher percentage of over $40\%$ for 'entry and not discovered $\phi$' case. Table 5.15 gives the number of tainted functions and arguments for Pseudo Random Number Generators rules.

| Type | # functions | # arguments |
|---|---|---|
| **R1** | 7 | - |
| **R2** | 7 | - |
| Other | 8 | - |
| **R1, R2** and other | 22 | 30 |

Table 5.15: Number of tainted functions and arguments for Pseudo Random Number Generators rules.

For rule **R1**, for discovering any static seed in a PRNG, our analysis discover $3,333 (14.79\%)$ firmware images to be violated in the case of 'entry and possible $\phi$' and $2,491 (11.06\%)$ firmware images for 'entry and not discovered $\phi$ case. The violations are found in two PRNG initialization functions, `srand()` and `srandom()`, where no initial seed is presented. Function `srandom()` stated that "If no seed value is provided, the random() function is automatically seeded with a value of 1." Thus, any results that we identify with no seeded value of the `srandom()` function argument are marked as a violation of rule **R1**. The same holds for the `srand()` function argument. In total, for the 'entry and possible $\phi$' case, we discovered more than $15$ thousand function references that are not seeded the initialization PRNG function arguments (i.e., seeded as $1$) where $6,038$ are from `srandom()` and $9,709$ are from `srand()` function. For the case of 'entry and not discovered $\phi$', the functions references are dropped significantly to $2,626$ for `srand()` and $193$ for `srandom()`. Bear in mind that there is a possibility of re-initializing the seed of the specific functions to a secure seed in a different execution path, thus, multiple stages of analysis must be performed for every possible violation to verify any cryptographic misuse.

Figure 5.21 depicts the percentage of firmware images that a cryptographic misuse for rule **R1** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year. We can observe that the firmware images that use static seed have significantly reduced over the years, with a peak of $48.08\%$ in the year in $2013$ and reaching $2.02\%$ in the latest complete year in $2020$. The year $2021$ is not complete and can be considered an outlier. The top $5$ discovered binaries that violate rule **R1** in the case of 'entry and not discovered $\phi$' for are: 'htpasswd', 'etm', 'login.cgi', 'cet', 'dispatcher.cgi' discovered a violation at least one time in $2376$, $95$, $84$, $45$ and $36$ firmware images respectively. The binaries are found in multiple vendors, including NETGEAR, Xiaomi, Zyxel, TP-Link and D-Link. Similar and additional executable binaries from the Ubiquiti vendor are discovered for the case of 'entry and possible $\phi$'. Some are called 'switchover', 'mini', 'ripened', and 'nsm', which are probably proprietary binaries as we speculate by their name. The overall depicted executable binaries names, along with the number of firmware images, are presented in Appendix Tables A.52 and A.56 in the case of 'entry and possible $\phi$' and in the case of 'entry and not discovered $\phi$' respectively.

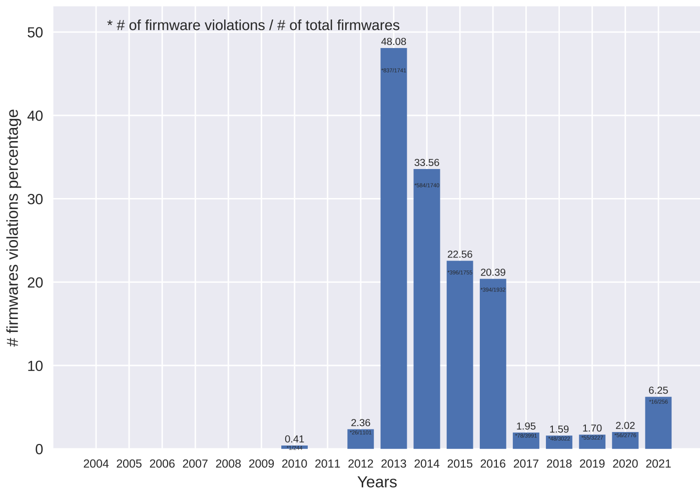



Figure 5.21: Discovered Cryptographic misuses over the years for rule **R1** for 'entry and not discovered $\phi$' case.

Regarding rule **R2** that is looking for any low entropy seeded random functions, we marked a violation of this rule when it is seeded with any combination of a result with two particular function calls as follows:

1. `getpid()` function call that returns the process ID of the calling process.

2. `time()` function call that returns the time as the number of seconds since the epoch (the number of seconds that have elapsed since *01/01/1970*).

The function call `time()` was the most popular among the two, with more than 117 thousand times called in an `srand() or srandom() or srand48()` function call. The `getpid()` function call is discovered more rarely compared to the other one, approximately 14 thousand times over all examined binaries. Overall, both cases are nearly similar and EBAT discovered more than 11 thousand firmware images, a percentage of over 50% for violating this rule, as half of our examined firmware images are using at least one time a seeded PRNG with any combination of `time()` and `getpid()` function calls. It should be noted that the context of the random functions is unknown, and there is a possibility not to be used in a security-related context. Additionally, there is the possibility of re-initialization with a secure seed. Thus, multiple stages of analysis for each case must be performed to verify any violations.

Figure 5.22 depicts the percentage of firmware images that a cryptographic misuse for rule **R2** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year. We can observe that the firmware images that use a low entropy source have increased over the years; the year 2021 is incomplete and can be considered an outlier. The top 7 discovered binaries for violating rule in the case of 'entry and not discovered $\phi$' for rule **R2** are: 'readyNASVault', 'zebra', 'htdbm', 'lighttpd', 'mysqlmanager' and 'cloudbrd' discovered a violation at least one time in 1790, 1010, 911, 899, 738, 714 and 628 firmware images respectively—the binaries found in multiple vendors including NETGEAR, Zyxel, ASUS and D-Link. Multiple proprietary executable binaries are also discovered from all vendors, such as 'LiveviewControlServer', 'EmbedThunderManager', and 'zytr069main'. The binaries that will be examined in the case study that results in a violation of rule **R2** and presented in section 5.7.2.3 of this chapter are the following: 'cgibin' from D-Link and 'HTTPd' from ASUS vendor. The overall depicted executable binaries names, along with the number of firmware images, are presented in Appendix Tables A.52 and A.56 in the case of 'entry and possible $\phi$' and in the case of 'entry and not discovered $\phi$' respectively.

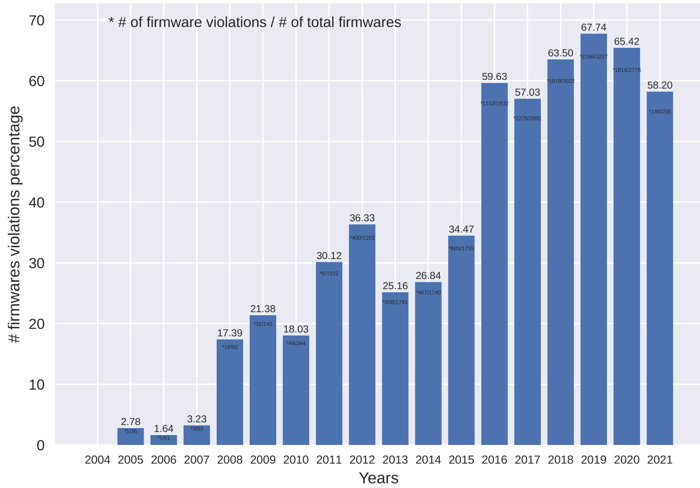



Figure 5.22: Discovered Cryptographic misuses over the years for rule **R2** for 'entry and not discovered $\phi$' case.

#### *5.7.1.4 Cryptographic Misuses in Key Derivation Functions (KDFs) and Password Based Encryption (PBE) rules*

Key Derivation Functions (KDFs) and Password Based Encryption (PBE) rules are discovered not to violate any of the four rules **K1, K2, K3** and **K4** in more than 13,000 (60%) of the examined firmware images in the case of 'entry and possible $\phi$ node'. In the case of 'entry and not discovered $\phi$ node', more than 15 thousand successfully unpacked firmware images are discovered not to violate any of the four rules. Table 5.16 depicts the total tainted functions for Key Derivation Functions (KDFs) and Password Based Encryption (PBE) rules **K1, K2, K3, K4** and the total ones. Total tainted arguments are also reported. Other functions present the tainted rules that helped us detect a rule or provide additional context to the rule we examined, such as the underlying

| Type | # functions | # arguments |
|---|---|---|
| **K1** | 48 | - |
| **K2** | 40 | - |
| **K3** | 20 | - |
| **K4** | 48 | - |
| Other | 77 | - |
| **K1, K2, K3, K4** and other | 233 | 579 |

Table 5.16: Number of tainted functions and arguments for Key Derivation Functions (KDFs) and Password Based Encryption (PBE) rules.

cryptographic hash function for detecting if rule **K4** is using a weak one or not. Only the total function arguments for all the examined rules and other functions are presented since there is no need to report the arguments separately for specific rules as they are dependent on other arguments for detecting a violation but also for detecting any additional context. In the following paragraphs, we examined each rule one by one.

| Function name (library) | # references | # binaries | # firmwares |
|---|---|---|---|
| `crypt` (libc, glibc) | 2,502 | 334 | 809 |
| `EVP_BytesToKey` (OpenSSL) | 140 | 20 | 66 |
| `Check_NAS_User_Password` (wrapper) | 70 | 6 | 23 |
| `Check_NAS_Administrator_Password` (wrapper) | 12 | 1 | 6 |

Table 5.17: Discovered violated functions for rule **K1** for 'entry and possible $\phi$ node'.

Specifically, Rule **K1** discovers the constant/fixed passwords used in KDF/PBE functions. In over $900$ firmware images, a constant fixed password is discovered for the 'entry and possible $\phi$ node' case. In addition, only $316$ firmware images are discovered without a $\phi$ node alternative. Table 5.17 presents the discovered function calls that violate rule **K1** in all firmware's binaries. Wrapper functions are also discovered and presented in the aforementioned table. The total unique binaries for the specific function are also presented in the 3rd column. The number of firmware images that violate this function is in the 4th column. The uniqueness of binaries is generated by a SHA-256 digest. Function call `crypt()` is the most widely used among the violated firmware images found in $334$ unique binaries and $809$ firmware images. Note that in each firmware image, more than one function call may be presented in the above table and possibly more than one binary can violate this rule. The most dominant discovered passwords are '`admin`', '`test`', '`this_is_a_passphrase`' and '`password`', where are some passwords that we cannot disclose them as they are unique and may compromise the device's overall security.

Figure 5.23 depicts the percentage of firmware images that a cryptographic misuse for rule **K1** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year. The year $2021$ is incomplete and can be considered an outlier as our dataset ends in the early second quarter of $2021$. We can observe from the figure that discovering a constant password embedded in firmware images remains relatively steady over the years, at approximately $1.5\%$. Some of the executable binaries names that a violation is discovered in the case of 'entry and not discovered $\phi$ node' for rule **K1** are the following: 'smm', 'eurl', 'qcmap_auth', 'change_password.cgi', 'daemon_fsp_app', 'pure-pw', 'commander' and 'authLogin.cgi', which are discovered in $148$, $45$, $36$, $23$, $16$, $15$, $9$ and $6$ firmware images, respectively. The exact context of those binaries needs further analysis. Some of them are probably proprietary executable binaries. The aforementioned executable binaries belong to the following vendors: D-Link, Ubiquiti, Linksys, TP-Link, NETGEAR and QNAP. The overall depicted executable binaries names, along with the number of firmware images, are presented in appendix tables A.53 and A.57 in the case of 'entry and possible $\phi$' and in the case of 'entry and not discovered $\phi$' respectively.

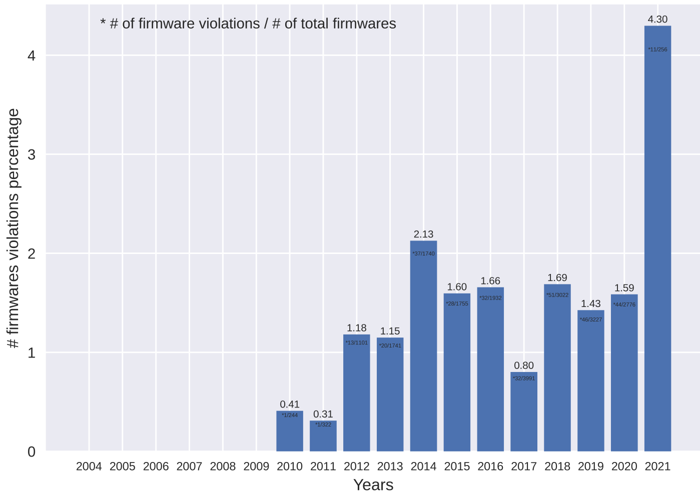



Figure 5.23: Discovered Cryptographic misuses over the years for rule **K1** for 'entry and not discovered $\phi$' case.

Rule **K2** examined the constant salts, or the absence of salt when using a PBE function. In total, $3,173$ $(14.07\%)$ and $1,901$ $(8.43\%)$ firmware images are discovered to

| Function name (library) | # references | # binaries | # firmwares |
|---|---|---|---|
| `crypt` (libc, glibc) | 10,158 | 1,659 | 3,112 |
| `EVP_BytesToKey` (OpenSSL) | 122 | 18 | 61 |

Table 5.18: Discovered violated functions for rule **K2** for 'entry and possible $\phi$ node'.

violate rule **K2**, from ‘entry and possible $\phi$ node’ and from ‘entry and not a discovered $\phi$ node’, respectively. Table 5.18 presents all the function calls that are discovered using a constant salt that violate rule **K2** in all firmware’s binaries for the ‘entry and possible $\phi$ node’ case. The discovered unique binaries are also reported in the 3$^{rd}$ column, and the number of firmware images found to violate this rule in the 4$^{th}$ column. A firmware image may find a violation of rules more than once a time in its binaries. Repetitively, `crypt` function call is the most dominant one, and salt `'aa'`, `'$1$'` and `'$1$mldcsfp$'` are the most discovered ones in all examined firmware images.

Figure 5.24 depicts the percentage of firmware images that a cryptographic misuse for rule **K2** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year, in the case of ‘entry and not discovered $\phi$’. The year 2021 can be considered an outlier as our dataset ends in the early second quarter of 2021. We can observe from the figure that discovering a constant salt embedded in firmware images has fluctuated over the years. Some of the executable binaries names that a violation is discovered in the case of ‘entry and not discovered $\phi$ node’ for rule **K2** are the following: ‘uhttpd’, ‘smm’, ‘busybox’, ‘sslvpnConfig’, ‘rc’, ‘makepwd’, ‘eurl’ and ‘synorcvol’, which are discovered in 1319, 148, 138, 122, 69, 66, 45 and 37 firmware images, respectively. The exact context of those binaries needs further analysis. Some of them are probably proprietary executable binaries. The aforementioned executable binaries belong to the following vendors: D-Link, Zyxel, TP-Link, NETGEAR and Xiaomi. The overall depicted executable binaries names, along with the number of firmware images, are presented in appendix tables A.53 and A.57 in the case of ‘entry and possible $\phi$’ and in the case of ‘entry and not discovered $\phi$’ respectively.

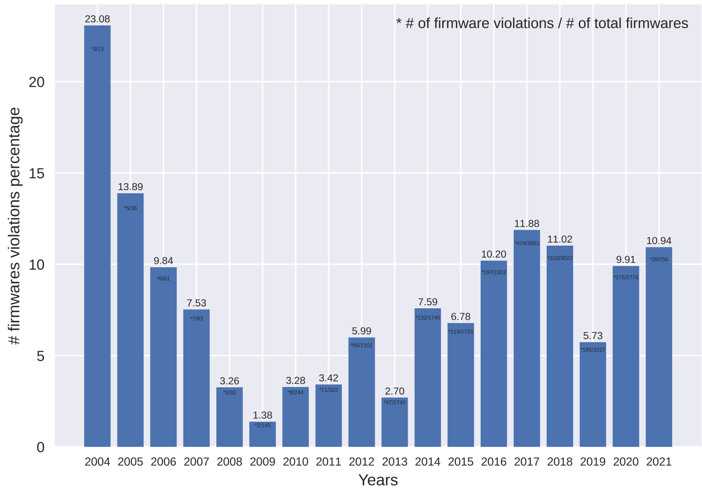



Figure 5.24: Discovered Cryptographic misuses over the years for rule **K2** for ‘entry and not discovered $\phi$’ case.

Figure 5.25 depicts the percentage of firmware images that a cryptographic misuse for rule **K3** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year, in the case of ‘entry and not discovered $\phi$’. Year 2021 can be considered an outlier as our dataset ends in the early second quarter of 2021. We can observe from the figure that the number of violations has slightly increased over the years, with a peak in the year 2017 to be 3.56%. Some of the executable binaries names that a violation is discovered in the case of ‘entry and not discovered $\phi$ node’ for rule **K3** are the following: ‘zycfgfilter’, ‘ubntbox’, ‘zcmd’, ‘ss-local’, ‘fw_printenv’, ‘ss-redir’, ‘eurl’ and ‘daemon_fsp_app’, which are discovered in 90, 76, 74, 68, 62, 54, 45 and 16 firmware images, respectively. The exact context of those binaries needs further analysis, although we can speculate from the binary name that most of them are proprietary binaries. The aforementioned executable binaries belong to the following vendors: Zyxel, Ubiquiti, DrayTek, NETGEAR, Linksys, Trendnet and Xiaomi. The overall depicted executable binaries names, along with the number of firmware images,

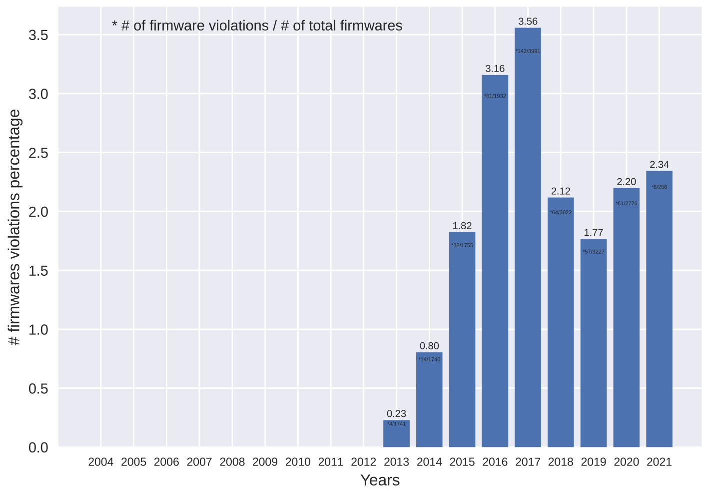



Figure 5.25: Discovered Cryptographic misuses over the years for rule **K3** for ‘entry and not discovered $\phi$’ case.

are presented in appendix tables A.53 and A.57 in the case of ‘entry and possible $\phi$’ and in the case of ‘entry and not discovered $\phi$’ respectively.

Table 5.19 depicts the overall discovered iterations found in all firmware’s binaries when examined for a violation of rule **K3**. Approximately $400$ firmware images violate this rule, as our threshold is set to below $1,000$ iterations in every cryptographic KDF/PBE function, similarly in both examined cases. We discovered more than $2$ thousand function calls belong to $2,122$ firmware images that used an iteration value equal to $1,000$, equal to the minimum threshold we defined in Chapter 2.3.5. Thus, no violation is marked. Note that the more times a KDF function is iterated, the longer it takes to compute the password hash. Therefore, the iteration count should be as large as the environment allows. Different devices may have a tolerance for a higher threshold than others. $4,096$ number of iterations are the most dominant ones, which provide fairly much better security than $1,000$, where $32,768$ number of iterations are also surprisingly used in more than $2$ thousand firmware images.

| Iterations | # Function calls | Details | # binaries | # firmwares |
|---|---|---|---|---|
| 32,768 | 2,066 | ’`gcry_kdf_derive`’ (GnuPG): 2,065, ’`PKCS5_PBKDF2_HMAC`’: 1 | 9 | 2,066 |
| 8,192 | 21 | ’`PKCS5_PBKDF2_HMAC_SHA1`’ (OpenSSL): 21 | 6 | 17 |
| 4,096 | 5,430 | ’`PKCS5_PBKDF2_HMAC_SHA1`’ (OpenSSL): 5,427, ’`wc_PBKDF2`’: 3 | 556 | 1,301 |
| 2,002 | 2 | ’`PKCS5_PBKDF2_HMAC`’ (OpenSSL): 2 | 2 | 2 |
| 2,000 | 17 | ’`PKCS5_PBKDF2_HMAC_SHA1`’ (OpenSSL): 17 | 8 | 17 |
| 1,024 | 3 | ’`PKCS5_PBKDF2_HMAC_SHA1`’ (OpenSSL): 3 | 1 | 3 |
| 1000 | 2,394 | ’`PKCS5_PBKDF2_HMAC_SHA1`’ (OpenSSL): 320, ’`gcry_kdf_derive`’ (GnuPG): 2,065, ’`PKCS5_PBKDF2_HMAC`’ (OpenSSL): 1, ’`EVP_BytesToKey`’ (GnuPG): 8 | 15 | 2,122 |
| *Marked as violated(**entry and possible** $\phi$)* | | | | |
| 5 | 608 | ’`EVP_BytesToKey`’: 608 | 102 | 306 |
| 2 | 64 | ’`EVP_BytesToKey`’: 64 | 2 | 16 |
| 1 | 378 | ’`EVP_BytesToKey`’: 378 | 31 | 306 |

Table 5.19: Discovered iterations in KDF/PBE function calls for all firmware binaries.

The underlying hash functions for KDF/PBE cryptographic rule **K4** are examined next. The rule is hard to examine as the underlying cryptographic function for `crypt()` and `crypt_r()` function calls depend on the *salt* that they are using. For instance, using a ‘`$5$`’ in front of the salt when it is passed as an argument on the `crypt()` function call means that a *SHA-256* encoded password algorithm will be used, which does not violate rule **K4**. On the other hand, the default KDF/PBE is based on *Data Encryption Standard (DES)*, and if ‘`$1$`’ is used, then it is based on *MD-5* which it does violate our rule **K4**. Thus, the above results are presented as `crypt()` is consistently violated if we do not discover the salt and we mark it as if it is using the default PBE based on *Data Encryption Standard (DES)*, which does not always hold. If we discover the salt, we map the start of the salt, e.g. ‘`$5$`’ - SHA-256, ‘`$1$`’ - MD5, to the discovered algorithm when reporting our results. The reader needs to keep that in mind if the salt cannot be discovered for `crypt()` and `crypt_r()` functions, then the default one is used (default on KDF/PBE is based on *Data Encryption Standard (DES)*) which violates rule **K4**.

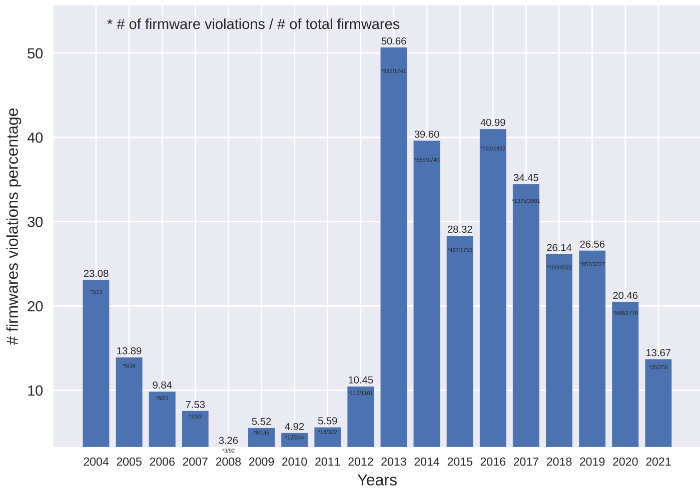



Figure 5.26: Discovered Cryptographic misuses over the years for rule **K4** for ‘entry and not discovered $\phi$’ case.

Figure 5.26 depicts the percentage of firmware images that a cryptographic misuse for rule **K4** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year, in the case of ‘entry and not discovered $\phi$’. Year $2021$ can be considered an outlier as our dataset ends in the early second quarter of $2021$. We can observe from the figure that the number of weak digests used in firmware images has reduced from year 2013 to our latest complete year 2020 with a peak to be 50.66% in 2013, which is dropped to 20.46% in year 2020. Some of the executable binaries names that a violation is discovered in the case of 'entry and possible $\phi$ node' for rule **K4** are the following: 'unix_chkpwd', 'unix_update', 'busybox', 'uhttpd', 'smm', 'getty', 'sslvpnConfig' and 'admin.cgi', which are discovered in 4835, 4648, 1523, 1319, 148, 128, 122 and 88 firmware images, respectively. The exact context of those binaries needs further analysis, and some proprietary binaries such as the 'admin.cgi' and 'basic_nis_auth' are discovered. The aforementioned executable binaries belong to multiple vendors, including Zyxel, Ubiquiti, NETGEAR, TP-Link, and ASUS. The overall depicted executable binaries names, along with the number of firmware images, are presented in Appendix Tables A.53 and A.57 in the case of 'entry and possible $\phi$' and in the case of 'entry and not discovered $\phi$', respectively.

The two functions that were discovered to violate rule **K4** are MD5 and PBE based on *Data Encryption Standard (DES)*. Table 5.20 depicts the total results for those functions. The most dominant discovered functions are `crypt` and `crypt_r`, which combined are the ones that result in nearly 90% of all detected violations on the firmware images. Bear in mind that `EBAT` marks them as violated if no constant salt is discovered; thus, multiple stages of analysis must be done individually to verify any possible violations.

| Function name (library) | Digest | # references | # binaries | # firmwares |
|---|---|---|---|---|
| *Marked as violated (**entry and not discovered $\phi$**)* | | | | |
| `crypt_r` (libc, glibc) | PBE base on DES | 45,847 | 68 | 4,616 |
| `crypt` (libc, glibc) | MD5 | 2,906 | 454 | 1,622 |
| `crypt` (libc, glibc) | PBE base on DES | 1,308 | 227 | 568 |
| `EVP_BytesToKey` (OpenSSL) | MD5 | 90 | 27 | 67 |
| *Marked as violated (**entry and possible $\phi$**)* | | | | |
| `crypt_r` (libc, glibc) | PBE base on DES | 45,847 | 68 | 4,616 |
| `crypt` (libc, glibc) | MD5 | 3,434 | 788 | 2,139 |
| `crypt` (libc, glibc) | PBE base on DES | 2,952 | 1,120 | 1,837 |
| `EVP_BytesToKey` (OpenSSL) | MD5 | 90 | 27 | 67 |

Table 5.20: Discovered violated functions for rule **K4**.

#### *5.7.1.5 Cryptographic Misuses in Message Authentication Codes (MACs) rules*

Message Authentication Codes (MACs) violations are discovered for the HMAC algorithm, where the number of firmware images that are not violating any of the three rules **M1, M2** and **M3** are found to be in 20,421 (90.57%) in the case of 'entry and possible $\phi$ node' and in 20,615 (91.43%) successfully unpacked firmware images in the case of 'entry and not discovered $\phi$ node'. Table 5.21 depicts the total tainted functions and function arguments for Message Authentication Codes (MACs) rules **M1, M2, M3** and the total ones. Other functions presented the tainted rules that helped us detect a rule or provide additional context to the rule we examined, such as the underlying cryptographic hash function for detecting if rule **M2** is using a weak one or not.

| Type | # functions | # arguments |
|---|---|---|
| **M1** | 16 | - |
| **M2** | 9 | - |
| **M3** | 16 | - |
| Other | 77 | - |
| **M1, M2, M3** and other | 118 | 417 |

Table 5.21: Number of tainted functions and arguments for Message Authentication Codes (MACs) rules.

Only the total function arguments for all the examined rules and other functions are presented since there is no need to report the arguments separately for specific rules as they are dependent on other arguments for detecting a violation but also for detecting any additional context. In the following paragraphs, we examined each rule one by one.

| Function name (library) | # references | # binaries | # firmwares |
|---|---|---|---|
| `HMAC` (OpenSSL) | 367 | 138 | 188 |
| `HMAC_Init_ex` (OpenSSL) | 288 | 9 | 144 |

Table 5.22: Discovered violated functions for rule **M1** for 'entry and possible $\phi$ node'.

The violation of rule **M1** for discovery constant encryption/decryption keys on MACs is discovered only in a small subset of firmware images, in 332 (1.47%) and 275 (1.22%) for 'entry and possible $\phi$ node' case and for 'entry and not a discovered $\phi$ node' case, respectively. We discovered only two unique keys used in multiple binaries, which cannot be disclosed as they are not publicly available, and we can compromise the device's overall security. Table 5.22 presents the two functions that belong to the OpenSSL library that the violations of this rule are discovered, along with the unique number of binaries and total number of firmware images.

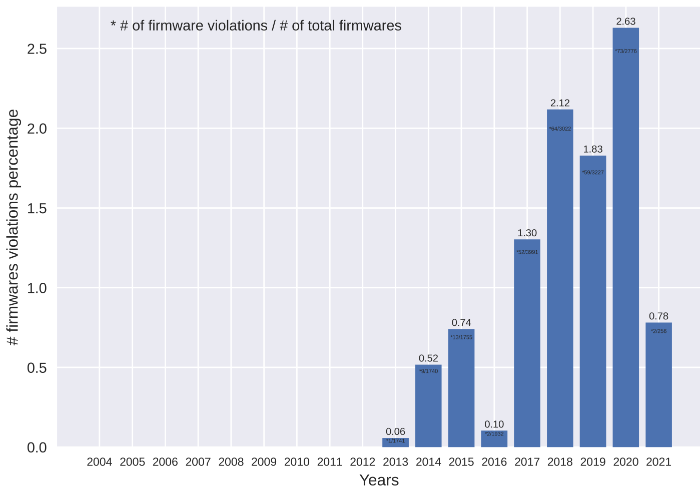



Figure 5.27: Discovered Cryptographic misuses over the years for rule **M1** for 'entry and not discovered $\phi$' case.

Figure 5.27 depicts the percentage of firmware images that a cryptographic misuse for rule **M1** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year, in the case of 'entry and not discovered $\phi$'. We can observe from the figure that the unique constant keys increased slightly over the years in percentage terms. The executable binaries names that this violation is discovered in the case of 'entry and not discovered $\phi$ node' for rule **M1** are the following: 'tr069_client', 'Netgear_ddns', 'httpd', 'ntgrddns' and 'pure-pw' which are discovered in $150$, $144$, $24$ and $14$ firmware images, respectively. The exact context of those binaries needs further analysis. Some of them are probably proprietary executable binaries. The first executable binary belongs to Draytek, and the others to the NETGEAR vendor. The overall depicted executable binaries names, along with the number of firmware images, are presented in Appendix Tables A.54 and A.58 in the case of 'entry and possible $\phi$' and in the case of 'entry and not discovered $\phi$', respectively.

| Function name (library) | Digest | # references | # binaries | # firmwares |
|---|---|---|---|---|
| *Marked as violated(**entry and not discovered** $\phi$)* | | | | |
| `HMAC` | MD-5 | 588 | 72 | 293 |
| `HMAC_Final` | MD-4 | 94 | 25 | 31 |
| `HMAC_Final` | MD-5 | 2,779 | 381 | 1,037 |
| `HMAC_Init` | MD-5 | 911 | 242 | 485 |
| `HMAC_Init_ex` | MD-4 | 94 | 25 | 31 |
| `HMAC_Init_ex` | MD-5 | 3,268 | 417 | 1,147 |
| `HMAC_Update` | MD-4 | 122 | 25 | 31 |
| `HMAC_Update` | MD-5 | 2,909 | 385 | 1,039 |

Table 5.23: Discovered violated functions and underlying cryptographic hash function for rule **M2**.

The most dominant discovered 'weak' underlying hash function on MACs that violates rule **M2** is *MD-5* cryptographic hash function, which is still in use for more than $1.5$ (over $7\%$) thousand examined successfully unpacked firmware images. Overall, $1,800$ $(7.98\%)$ firmware images are found to violate rule **M2** in the case of 'entry and possible $\phi$ node' and for the case of 'entry and not a discovered $\phi$ node' $1,646$ $(7.30\%)$ firmware images are discovered. OpenSSL HMAC functions are the ones discovered to violate our results. However, other functions from other libraries are also tainted, but no violation is discovered from entry. Table 5.23 presents all the discovered violated functions, unique binaries and firmware images for rule **M2**. All the presented functions belong to the OpenSSL cryptographic library. *MD-5* cryptographic hash function is the most used, whereas *MD-4* use is surprisingly discovered.

Figure 5.28 depicts the percentage of firmware images that a cryptographic misuse for rule **M2** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year, in the case of 'entry and not discovered $\phi$'. Year $2021$ is incomplete and can be considered an outlier as our dataset ends in the early second quarter of $2021$. We can observe from the figure that the discovery of weak digests increased over the years. Some of the executable binaries names that a violation is discovered in the case of 'entry and not discovered $\phi$ node' for rule **M2** are the following: 'wpa_supplicant', 'hostapd', 'ipsec', 'prog-cgi', 'daemon_fsp_app', 'snmpd', 'dhclient' and 'mdb', which are discovered in $625$, $389$, $350$, $297$, $126$, $79$, $71$ and $60$ firmware images, respectively. The exact context of those binaries needs further analysis. However, some of the binaries are recognized by their name. The aforementioned executable binaries belong to the following vendors: MicroTik, Ubiquiti, Synology, ASUS, NETGEAR and D-Link. The overall depicted executable binaries names, along with the number of firmware images, are presented in Appendix Tables A.54 and A.58 in the case of 'entry and possible $\phi$' and in the case of 'entry and not discovered $\phi$', respectively.

Lastly, rule **M3** discovers the non-secure key length used in MAC functions. A non-secure key length is discovered only for $80$ firmware images for the 'entry and possible $\phi$ node' case. In addition, only $79$ firmware images are discovered without a $\phi$ node alternative. Figure 5.29 depicts the percentage of firmware images that a cryptographic misuse for rule **M3** is discovered (violated) over multiple years in our dataset, calculated over the violated firmware images divided by the total successfully unpacked firmware images for each year, in the case of 'entry and not discovered $\phi$'. We can observe from the figure that the most violations for rule **M3** occur in the year $2020$, to be $1.44\%$, where in previous years remains in low percentage and most of them at zero. The executable binaries names that a violation is discovered in the case of 'entry and not discovered $\phi$ node' for rule **M3** are the following: 'hostapd', 'dimclient', 'wpad' and 'tincd', which are discovered in $40, 31, 8$ and $1$ firmware images, respectively.

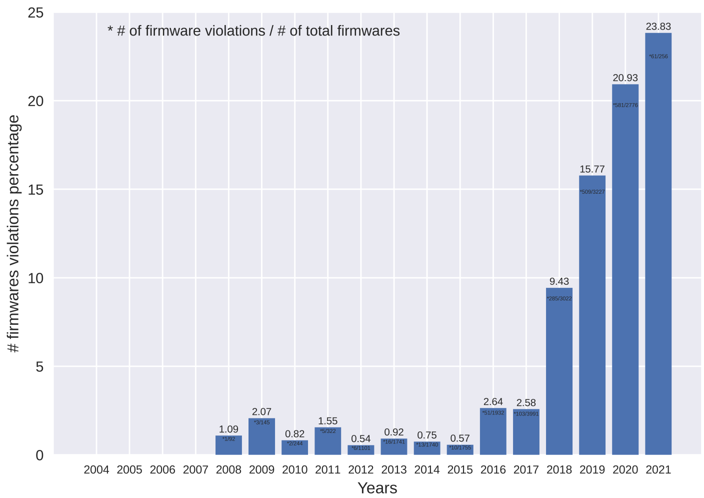



Figure 5.28: Discovered Cryptographic misuses over the years for rule **M2** for 'entry and not discovered $\phi$' case.

The exact context of those binaries needs further analysis. The aforementioned executable binaries belong to the following vendors: ASUS, Alfa, Linksys, TP-Link, NETGEAR, Synology and Tenda. The overall depicted executable binaries names, along with the number of firmware images, separately for each vendor, are presented in Appendix Tables A.54 and A.58 in the case of 'entry and possible $\phi$' and in the case of 'entry and not discovered $\phi$', respectively.

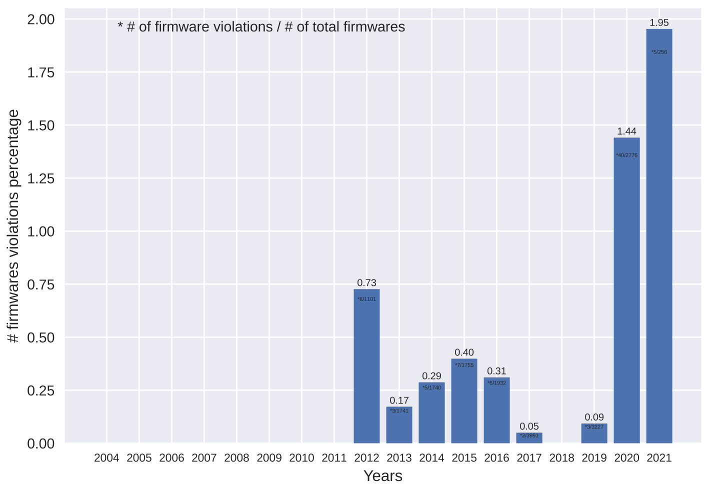



Figure 5.29: Discovered Cryptographic misuses over the years for rule **M3** for 'entry and not discovered $\phi$' case.

Table 5.24 represents the overall discovered key length in MACs function calls for all firmware's binaries. The 1st column represents the key length in bytes, and the 2nd column shows how many function calls are discovered in all of our binaries. The 3rd column presents more details of the discovered functions and the number of function calls. Functions `HMAC_Init_ex()` and `HMAC_Init()` are the ones that violate rule **M3** with $1$, $3$ and $8$ bytes of key length respectively. Only a tiny percentage (less than $0.5\%$) of firmware images are discovered to violate this rule. All the functions mentioned in Table 5.24 are from OpenSSL cryptographic library except `'gcry_md_setkey'` from GnuPG, `'wc_HmacSetKey'` from WolfSSL. `EBAT` also discovers three wrapper functions named `'csrComputeHMACSHA256'`, `'fr_hmac_md5'` and `'hmac_hex'`.

#### *5.7.1.6 Cryptographic Misuses in Authenticated encryption/decryption and AEAD rules*

Lastly, regarding Authenticated encryption/decryption and AEAD rules **A1** and **A2**, we do not discover any violation in all examined successfully unpacked firmware images. Table 5.25 depicts the total tainted functions for Authenticated encryption/decryption and AEAD rules **A1** and **A2** and the total ones. Total tainted arguments are also reported. Other functions present

| Type | # functions | # arguments |
|---|---|---|
| **A1** | 16 | - |
| **A2** | 9 | - |
| Other | 18 | - |
| **A1, A2** and other | 47 | 193 |

Table 5.25: Number of tainted functions and arguments for Authenticated encryption/decryption and AEAD rules.

| # bytes | # Function calls | Details | # binaries | # firmwares |
|---|---|---|---|---|
| 8 | 110 | not discovered from entry; thus, not marked as a violation `'HMAC_Init_ex'`: 96, `'csrComputeHMACSHA256'`: 14 | 20 | 39 |
| 16 | 8,593 | `'HMAC'`: 4613, `'HMAC_Init_ex'`: 2767, `'HMAC_Init'`: 850, `'fr_hmac_md5'`: 216, `'csrComputeHMACSHA256'`: 77, `'gcry_md_setkey'`: 70 | 855 | 3,180 |
| 20 | 2,823 | `'HMAC'`: 1785, `'HMAC_Init_ex'`: 984, `'gcry_md_setkey'`: 54 | 299 | 674 |
| 22 | 4 | `'HMAC_Init_ex'`: 4 | 4 | 2 |
| 23 | 9 | `'HMAC'`: 9 | 1 | 3 |
| 24 | 24 | `'HMAC_Init_ex'`: 24 | 1 | 8 |
| 30 | 3,004 | `'HMAC_Init_ex'`: 74, `'gcry_md_setkey'`: 2930 | 27 | 1,377 |
| 32 | 21,747 | `'HMAC'`: 6798, `'HMAC_Init_ex'`: 8021, `'HMAC_Init'`: 948, `'hmac'`: 60, `'hmac_hex'`: 15, `'csrComputeHMACSHA256'`: 7, `'gcry_md_setkey'`: 5895, `'wc_HmacSetKey'`: 3 | 1,289 | 7,359 |
| 33 | 12 | `'HMAC'`: 12 | 6 | 6 |
| 36 | 1,557 | `'HMAC'`: 44, `'HMAC_Init_ex'`: 48, `'gcry_md_setkey'`: 1465 | 32 | 1,383 |
| 48 | 56 | `'HMAC_Init_ex'`: 56 | 9 | 23 |
| 52 | 376 | `'HMAC'`: 376 | 9 | 188 |
| 62 | 1,503 | `'HMAC_Init_ex'`: 38, `'gcry_md_setkey'`: 1465 | 13 | 1,361 |
| 64 | 1,212 | `'HMAC'`: 19, `'HMAC_Init_ex'`: 186, `'HMAC_Init'`: 41, `'gcry_md_setkey'`: 966 | 48 | 1,158 |
| 68 | 1,503 | `'HMAC_Init_ex'`: 38, `'gcry_md_setkey'`: 1465 | 13 | 1,361 |
| 108 | 42 | `'HMAC_Init'`: 42 | 8 | 14 |
| 128 | 3,674 | `'HMAC_Init_ex'`: 744, `'gcry_md_setkey'`: 2930 | 108 | 1,705 |
| 160 | 27 | `'HMAC'`: 27 | 1 | 9 |
| | | *Marked as violated(**entry and possible** $\phi$)* | - | - |
| 1 | 1 | `'HMAC_Init'`: 1 | 1 | 1 |
| 3 | 62 | `'HMAC_Init'`: 62 | 8 | 31 |
| 8 | 96 | `'HMAC_Init_ex'`: 96 | 26 | 48 |

Table 5.24: Discovered key length in Message Authentication Codes (MACs) function calls for all firmware's binaries.

the tainted rules that helped us detect a rule or provide additional context to the rule we examined, such as a key/iv *length* to perform an authenticated key encryption/decryption. One reason we do not discover any violations may lie in the narrow rules we examined for rules **A1** and **A2** as depicted in the aforementioned table. Adding more rules will further expand these rules and possibly discover any violations, if any, are present.

## 5.7.2. Case Studies

Multiple case studies evaluate `EBAT's` ability to detect cryptographic misuses are presented in this subsection. Those case studies are real-world cases used to test the effectiveness of our implemented tool, the limitations, and the possibilities for improvements. Each case study takes considerable time to evaluate; thus, a limited subset of those are presented. Firstly, the unpacked module is examined, and the ability to add a decrypted unpacking module for specific firmware image types is presented, providing valuable findings. Furthermore, a case study of a recent high severity `CVE` in a *TP-Link* firmware image that uses a hard-coded cryptographic key discovered in a cryptographic function call, noted as a cryptographic misuse rule, is evaluated, comparing with `EBAT's` ability to discover it. In addition, a CVE reported for predictable seed in Pseudo-Random Number Generator (PRNG) is also examined and evaluated with `EBAT` ability to discover it. Lastly, `CryptoREX` related paper is compared with our `EBAT` taint analysis module, where the findings are compared.

### *5.7.2.1 Firmware Decrypt module in D-Link firmware images*

The first step in a successful firmware image security analysis is unpacking. `EBAT` encountered many firmware images that were unable to unpack successfully and optimized from time to time to include more unpacking modules and methods to unpack more and more firmware images successfully. Due to the large-scale analysis, we encountered a few products that `EBAT` failed to unpack successfully (mainly due to encryption); however, in the initial firmware release, `EBAT` managed to unpack them. Further investigation in *DIR's* product line from `D-Link` we came across a report at [25], where researcher(s) manage to break the encryption of the encrypted firmware image with the physical acquisition of the device and manage to extract the so-called `'imgdecrypt'` executable binary that is responsible for the decryption of every firmware image in the particular product without the physical acquisition of the device. Fortunately, performing a large-scale analysis covering all firmware image releases across a product's life span, we discover some initial releases from *D-Link DIR's* product lines that are not encrypted, either part of it or as a whole; thus, EBAT successfully unpacks them, and we are able to locate the aforementioned executable binary called 'imgdecrypt'.

EBAT's analysis discovers cryptographic misuses on the 'imgdecrypt' executable binary in less than a few minutes. Briefly, EBAT reports that the binary uses the *OpenSSL* cryptographic library and performing the static taint analysis module it manages to discover the AES decryption constant key from 'AES_set_decrypt_key()' function that violates *Rule S1*. Additionally, it discovered the key length, which is 128 bits, the mode of encryption, *CBC*, the IV length, which is 16 bytes and also discovered the *RSA and SHA-512* digest functions that are being used. EBAT analysis also reports the usage of constant input (plaintext) in decryption/encryption functions and discovered the usage of constant plaintext input in the AES_cbc_encrypt function. For a full analysis report on why all of these are indications of cryptographic misuses, more details are given at [25], where the authors also similarly implemented a decrypting module.

Overall, the protection of the encrypted firmware image module that *DIR's D-Link products line* is using has a severe security flaw that comes with the violation of using a constant encrypted key found embedded in the binary. All of that could be prevented by using EBAT as an analyzing security tool to scan their firmware image before releasing the product and patching any issues. The firmware decrypt module first reported in [25] is also implemented in our unpacking module code and deployed with our tool when running on a large scale. Using the *D-Link - DIR* decrypt implemented module, we additionally discover multiple products and product lines that are affected, not only on *DIR's* product lines but *DAP's* and COVR's powerlines as well. Our large-scale report discovered the following:

- More than 10 products from *DIR, COVR, DAP* product lines are discovered to use the vulnerable 'imgdecrypt' executable binary.
- In total 66 firmware images from 2017 until their latest capture release version (04/2021 is the latest capture firmware image executed by our implemented crawler) were found to have the vulnerable 'imgdecrypt' executable binary embedded in the firmware image.

#### *5.7.2.2 Hard-coded Cryptographic Key in TP-Link firmware image*

A recent *high* severity CVE-2020-10884[12], with score of *8.8* is published for TP-Link Archer A7 Firmware AC1750 router, firmware Version: 190726[13]. The vulnerability was reported to the vendor on *19/11/2019* and taken public on *25/03/2020* from Zero Day Initiative[14]. The vulnerability results from using a hard-coded encryption key, which an attacker can leverage in conjunction with other vulnerabilities to execute code in the context of root. In addition, authentication is not required to exploit this vulnerability.

The specific firmware image is also analyzed by EBAT as it is found to be in our publicly crawled dataset. The firmware image analysis finished in approximately 15 minutes running on a personal computer with Ubuntu OS (CPU i7-8400, 16 Gbytes of RAM) using multiple threads. The hard-coded credential is also discovered using static taint analysis from EBAT's module that results in a violation of *Rule S1: "Usage of constant encryption/decryption keys for various block and stream ciphers (Cryptographic Misuse Rule)"*. Specifically, the static key and IV using AES [75] encryption algorithm are reported at [85]. EBAT analysis successfully discovered the fixed key[15] given also the underlying cryptographic algorithm (AES 128 bit key with CBC mode of operation). Additionally, as mentioned in the report, the key that they have constantly embedded in the binary is 256 bits long, but only 128 bits are in use. EBAT successfully recovered only the 128 bits that are in use as it discovered the underlying cryptographic algorithm key length. Unfortunately, the fixed IV is not discovered and missed by EBAT, although it is manually verified to be there and left for future work as an improvement of our tool. Overall, in approximately 15 minutes of automated analysis, one can address the issue way earlier just by using EBAT analysis and informing the developers before releasing any of the firmware images.

A large-scale analysis offers the ability to search the database for similar occurrences of the same vulnerability. Firstly, we search for any occurrences of the vulnerable binary[16] and unfortunately we

[12] NVD - CVE-2020-10884, National Vulnerability Database, 2020, https://nvd.nist.gov/vuln/detail/CVE-2020-10884

[13] Download for Archer A7 | TP-Link, TP-Link, 2020, https://www.tp-link.com/us/support/download/archer-a7/

[14] (Pwn2Own) TP-Link Archer A7 'tdpServer' Use of Hard-coded Cryptographic Key Remote Code Execution Vulnerability, ZDI-20-336 | Zero Day Initiative, 2020, https://www.zerodayinitiative.com/advisories/ZDI-20-336/

[15] Symmetric constant key in base64 'VFBPTkVNRVNIX0tmIXhuPw==' and ASCII: 'TPONEMESH_Kf!xn¿

[16] 'tpdServer' binary SHA-256 digest 7409588ca41d469e1485fa3e6a48cee772fffaa2adadb03e29dc878a1c032a32

found that one firmware image from a different product had the same exact binary, hence an identical vulnerability on 'tdpServer' executable binary that is patched in a later release version as well. Furthermore, we scan the TP-Link database for occurrences of the same unique AES symmetric key and the query results in 68 different firmware images from 24 different products, including the TP-Link Archer A7, other TP-Link's routers and TP-Link's Wi-Fi Range Extenders (mesh and not). Interestingly, some of the newly discovered possibly vulnerable firmware images come with release dates of early 2021, a year after the CVE-2020-10884 is disclosed. We decided not to disclose any of the firmware images and products, as some of them may be vulnerable to *n-day* attacks.

#### 5.7.2.3 Predictable seed in Pseudo-Random Number Generator

EBAT cryptographic misuse analysis checks for the usage of weak seeds, used in Pseudo-Random Number generators (PRNGs), **Rule R2**, from Chapter 2.3. In order to verify the use of it, we are trying to identify if we successfully discovered the *high* severity CVE-2020-13784[17] with a base score of 7.5 on *D-Link DIR-865L Ax 1.20B01 Beta devices*, that states to have a predictable seed in a Pseudo-Random Number Generator (PRNG).

Analyzing the aforementioned firmware image in approximately 20 minutes, EBAT discovers the executable binary called 'cgibin' that violates **Rule R2** in a function called 'get_random_string' that uses 'srand(time)' and 'rand()' functions to produce a pseudo-random sequence. Time is a predictable seed and should not be used. Verify our results with the report at [30], which shows that the generator is used to generate a random session cookie. However, an attacker who knows the time of the request can predict it and determine the session cookie to conduct multiple attacks. Overall, with EBAT analysis, the developers can easily spot the 'random string generator' not to be so random and patch it appropriately. Unfortunately, scanning our database for similar instances of executable binary 'cgibin' that violates **Rule R2** from a function called 'get_random_string', results in a hit of 23 firmware images, including the one reported, and 8 different products varies between routers and WiFi range extenders. The last possibly vulnerable firmware image is released in the third quarter of 2020. The complete analysis report will not be disclosed as there is also a possibility of *n-day* attacks on those devices (products).

Another high severity CVE-2017-15654[18] with a base score of 8.3 on highly predictable session tokens in the 'HTTPd' server in all current versions (<= 3.0.0.4.380.7743) of ASUS software ASUSWRT allows an attacker to gain administrative router access. EBAT analysis is on a binary level and not in open-source code, where the specific source code is compiled on a binary that we analyze with static taint analysis to detect cryptographic misuses. The predictable pseudo-random generator is fully disclosed at [26], where the function named 'generate_token' is used to generate the session token for an authenticated user using *stdlib* rand function with a weak entropy as seed to be srand(time(NULL)). Scanning EBAT results database for violation of *Rule R2* in 'HTTPd' binary 'generate_token' results in 71 firmware images before the published date of CVE-2017-15654 ranging from 2015 to 2017. In our dataset, a total of 29 devices (products) with this vulnerable firmware image are affected. No new firmware images after the CVE has been patched are found. Overall, if EBAT analysis is used in a firmware image by 2015, the aforementioned CVE will be patched way earlier and not be actively exploitable in a period of 2 years.

#### 5.7.2.4 CryptoREX comparison

CryptoREX presented in [102] is a framework to identify cryptographic misuse of *IoT/embedded* devices. CryptoREX is executed on 521 successfully unpacked firmware images over 1,327 crawled firmware images with 165 pre-defined crypto APIs. Compared with EBAT where it is executed on 22,548 successfully unpacked firmware images (including 424 partially unpacked firmware image) over 36,073 crawled firmware images, with 733 pre-defined crypto APIs. The large-scale analysis of this study is not directly comparable with CryptoREX study as the examined firmware images are 43 times greater. Additionally, the dataset of firmware images used by CryptoREX is not by all means included in EBAT's dataset, as we do not have the overall information on the examined dataset directly from the related paper [102]. The pre-defined crypto APIs from CryptoREX are included in our list and expanded by

[17] NVD - CVE-2020-13784, National Vulnerability Database, 2020, https://nvd.nist.gov/vuln/detail/CVE-2020-13784

[18] NVD - CVE-2017-15654, National Vulnerability Database, 2017, https://nvd.nist.gov/vuln/detail/CVE-2017-15654

us. We also covered and expanded the cryptographic rules in this study. EBAT taint analysis module is inspired by CryptoREX. However, the implemented code/tools are completely different. CryptoREX uses Valgrind's VEX IR [69] as the representation format; its Python bindings PyVEX [89] using Angr [1]. On the other hand, we use Ghidra SRE [70], *Ghidra's* intermediate representation/language (IR/IL) form, and Ghidra's *P-Code*. After the IR form, CryptoREX implements their taint analysis based on Angr [1] and IDA Pro [49] (not open source available) in order to enhance their analysis with multiple techniques. In our implementation, we rely on Ghidra's analysis and enhancement tools, and where applicable, we developed headless scripts for further enhancement. We also developed the taint analysis headless script for detecting cryptographic violations. Despite the tools and enchantments, there are similarities in both implementations, such as the Cross-file Call Graph construction for the detection of library wrappers. CryptoREX has also implemented a module to simulate the functionality of array operation APIs (e.g., memset() and memcpy()), where we did not implement an extensive module to simulate all the operations and left it for future work. EBAT implements a way to monitor the cryptographic context of the function call that allows us to provide more details about the cryptographic primitives used in firmware images. In addition, there are other modules that we implemented and presented in previous sections that are not related to taint analysis of cryptographic misuse detection. Overall, a direct comparison cannot be performed between CryptoREX and EBAT. ***However, EBAT covers the cryptographic rules of CryptoREX, we strongly suggest running CryptoREX as well as EBAT on any examined firmware image. Due to different analysis techniques, CryptoREX may discover violations that EBAT misses and vice versa.***

| Violated Rule | Identical Rules | CryptoREX | | EBAT (entry and not discovered $\phi$) | | EBAT (entry and possible $\phi$) | |
|---|---|---|---|---|---|---|---|
| | | # of Firm. | % of Firm. | # of Firm. | % of Firm. | # of Firm. | % of Firm. |
| Rule 1 | S2 | 107 | 20.5% | 3,794 | 16.83% | 3,794 | 16.83% |
| Rule 2 | S3 | 24 | 4.6% | 21 | 0.09% | 21 | 0.09% |
| Rule 3 | S1 and K1 | 59 | 11.3% | 674 | 2.99% | 674 | 3.01% |
| Rule 4 | K2 | 56 | 10.8% | 1,901 | 8.43% | 3,173 | 14.07% |
| Rule 5 | K3 | 23 | 4.4% | 441 | 1.96% | 445 | 1.97% |
| Rule 6 | R1 | 0 | 0% | 2,491 | 11.05% | 3,333 | 14.78% |
| No violation | - | 395 | 75.8% | 15,524 | 68.85% | 13,697 | 60.75% |

Table 5.26: Results of crypto misuse detection (by rules) as reported on [41] and CryptoREX [102] compared with EBAT.

Table 5.26 depicts the overall CryptoREX results of discovered cryptographic misuses as reported in [102] compared with EBAT results. As explained earlier, we cannot make a direct comparison in our results as the tools, implementations, techniques and, most importantly, the dataset differ. Thus, the results we presented in this section are the ones covered by CryptoREX and compared the identical results that are also covered by EBAT. Table 5.26 presents the results for EBAT in two ways. The findings from a rule violation from a binary's entry point and not discovered as a $\phi$ node, and the ones that are discovered from a binary's entry point and may possibly be a $\phi$ node. CryptoREX violation of rule $1$ is discovered percentage-wise close with EBAT despite the difference in firmware images. EBAT discovered the violation of rule $2$ only in $21$ firmware images, $4$ less than CryptoREX, however, there is a large data-set difference. Rule $3$ and $4$ are much higher in EBAT's discovery (percentage-wise), whereas rule $5$ has a low percentage but was discovered in more than $400$ firmware images. Additionally, the violation of rule $6$ is discovered in $2,491$ firmware images from EBAT in the case of 'entry and not discovered $\phi$'. However, remember that false positives may exist in our results as explained in the previous section 5.7.1.3 mainly due to context re-initialization of random function and the context of usage. The total number of 'no violations' is close to ours. However, if we exclude rule $6$ from EBAT results we have the following for no violation (excluding rule 6): $16,815$ firmware images ($74.57\%$) from 'entry and not a discovered $\phi$ node' case, and $15,698$ firmware images ($69.62\%$) from 'entry and possible $\phi$ node' case, discovered to not violating any of the above rules. CryptoREX discovered that $75.8\%$ of their total evaluated firmware images does not have a single discovered violation close to our results when excluding Rule 6. Without excluding Rule 6, the percentage is approximately $70\%$ for the 'entry and not discovered $\phi$' case and $60\%$ in the 'entry and possible $\phi$ node' case.

### 5.7.3. Conclusions and Validity of results

In the paragraphs below, the validity of the aforementioned results will be examined, along with our conclusions. Overall, EBAT executed in $22,548$ successfully unpacked firmware images and analyzed for cryptographic misuses $1,452,039$ binaries where $861,946$ are executables and the rest are libraries. In total, our evaluation of the results produced the following. In the case of 'entry and possible $\phi$' more than $10$ thousand firmware images (approximately $48\%$) discovered to non-violate at least one of the rules excepting rules ***R1*** and ***R2***, where in the cases of including all the rules the percentage drops to $29.44\%$ resulting in more than $6$ thousand successfully unpacked firmware images. In the case of 'entry and not discovered a $\phi$ node' $12,040$ $(53.40\%)$ firmware images are discovered to not violating any of the rules (except rule **R1** and R2), where if we do not exclude those rules the percentage drops to approximately $33\%$. Specifically, in the case of 'entry and not discovered $\phi$' we have the following: for the Symmetric Key Cryptography rules ***S1***, ***S2***, ***S3*** and ***S4*** we observe a non-violation of any of these rules to be at $79.34\%$ of the total successfully unpacked firmware images. In addition, for the Public Key Cryptography rules ***P1***, ***P2*** and ***P3*** is $79.70\%$ for not violating any of the public key cryptography rules. Furthermore, approximately $40\%$ are observed for Pseudo Random Number Generators (*PRNGs*) rules ***R1*** and ***R2*** to non-violate any of these rules, where the security-context is unknown. Regarding Key Derivation Functions (*KDFs*) and Password Based Encryption (*PBE*) rules ***K1***, ***K2***, ***K3*** and ***K4*** a total of $68.80\%$ are discovered to be non violating any of the rules. Lastly, Message Authentication Codes (*MACs*) rules ***M1***, ***M2*** and ***M3*** are found to non violating approximately $90\%$ of the total firmware images. Not a single violation is discovered for Authenticated encryption/decryption and *AEAD* rules ***A1*** and ***A2***.

The evaluation of the above results may come with false positives or incomplete results, and it is strictly noted that one should further verify any of `EBAT` results through **multiple stages of analysis**, as we show with the case studies in Section 5.7.2. The multiple stages of analysis on each specific firmware image include but are not limited to manual audit, dynamic analysis techniques, and the physical acquisition of a device for verifying any potential vulnerability that `EBAT` discovers. For instance, scenario listing 5.3 explains why manual audit, dynamic analysis and physical device acquisition are necessary to verify cryptographic misuses. However, our results still provide a first good indication of the security of the examined firmware images and the cryptographic weaknesses they may face.

The rules that are excluded for the total no violation results are ***R1*** and ***R2***, which may result in a variety of false positives due to many factors, as we explain in this paragraph. Specifically, the main problem is that we cannot automatically, using `EBAT`, determine the context of the random functions, which means that if they are used in a cryptographic application, for instance, a request of random numbers for the Diffie Hellman Key Exchange (DHKE), or used in non-cryptographic applications where Cryptographically-Secure Pseudorandom Number Generators (CSPRNGs) are not strictly necessary. Additionally, we cannot check individually all the function calls as there is a chance of re-initialization of the random seed, where a non-violated version (or a $\phi$ path) of the seeding of a PRNG may occur after the vulnerable one, and/or even the vulnerable function call may never occur in the case of a $\phi$ node. We cannot cover these cases in our tool; thus, we decided to present those results that were excluded from the total ones.

`EBAT` modules for static taint analysis depend on the taint functions and taint function's arguments; thus, the list provided is not by all means an exhaustive list. The modularity of `EBAT` provides the expansion, editing, or rewriting of the list of taint functions and arguments in anyone's needs. In addition, Ghidra's newer versions were also released when writing this thesis, with many improvements and bug fixes. The Ghidra headless scripts are backwards compatible, and newer versions may provide additional results and better precision. Our implementation code is open-source so that anyone can contribute to the project.

Several case studies are examined in the section mentioned above 5.7.2, where we examined previously known vulnerabilities for cryptographic misuses in multiple firmware images and using our implemented tool, `EBAT`, we verify the potential of automatic discovery of those vulnerabilities, that lead to a cryptographic misuse for different rules covered in this section. In addition, we discover more products that are not reported in the affected products using our large-scale analysis that is performed using `EBAT`. In conclusion, the results provide a good first indication of the security of the examined firmware images and the cryptographic weaknesses they may face; however, multiple stages of analysis is needed to verify any of the claims.

# 6 Conclusion

In today's interconnected world, the widespread adoption of the Internet of Things (IoT) and embedded devices has revolutionized various aspects of our lives. These devices, ranging from smart home appliances to industrial control systems, rely heavily on firmware images to provide essential functionalities and operations. The software security of the Internet of Things(IoT)/embedded devices primarily relies on their firmware images. However, with these devices' increasing complexity and diversity and the rapid pace of technological advancements, firmware image security has become more challenging to analyze. It is crucial to address the potential risks and vulnerabilities associated with firmware images where a vendor may prevent them by implementing secure development practices, effectively managing vulnerabilities, and providing regular security updates. In this research, we delve into the realm of firmware image security of IoT/embedded devices and aim to gain a deeper understanding of the security issues and potential risks faced by IoT/embedded devices related explicitly to their firmware images.

This thesis explores the security of firmware images in IoT/embedded devices. It implements an open-source tool called ***E**mbedded **B**inary **A**nalysis **T**ool*[1]. `EBAT` provides an automated and comprehensive security analysis of firmware images, identifying possible vulnerabilities and weaknesses. A large-scale analysis of diverse IoT/embedded devices demonstrates the effectiveness of `EBAT` in analyzing firmware security in various aspects. The large-scale analysis is conducted in more than 30,000 firmware images used by home users to corporate environments belonging to more than 5,000 IoT/embedded devices across 33 vendors in a date span of over 20 years. The results and findings obtained from this analysis have been presented in the preceding chapters, providing valuable insights into the state of firmware image security. In the rest of this section, `EBAT's` main contributions and a summary of the results will be presented, concluding with our final thoughts, limitations and future work.

## 6.1. EBAT Contributions

This thesis has presented the implementation and capabilities of the *Embedded Binary Analysis Tool* (`EBAT`) for analyzing the security of firmware images in IoT/embedded devices. Throughout this thesis, we have demonstrated the functionality and effectiveness of `EBAT` in addressing critical aspects of firmware security. Firstly, an automated process is implemented, utilizing a crawler to download an extensive amount of firmware images from numerous vendors for various types of IoT/embedded devices. The dataset obtained through this process is organized into multiple products, with each product containing publicly available firmware images arranged chronologically. The tool's automated process allows for analyzing multiple firmware images on a large scale. It offers an automated and comprehensive approach to assessing the security of these devices by providing valuable insights into possible vulnerabilities and weaknesses. The tool exclusively utilizes open-source software, enabling the implementation of a complex firmware security analysis tool without relying on any proprietary software. Moreover, several modules are implemented using Python scripts, Ghidra headless scripts, and various open-source libraries libraries. Overall, `EBAT` provides a comprehensive tool-set for analyzing the

[1] `EBAT` is provided open source at EBAT-public, `https://github.com/ppanagiotou/EBAT-public`

security of firmware images in IoT/embedded devices by leveraging its automated analysis capabilities and utilizing various modules; it offers insights into the presence of security vulnerabilities, discovered any lack of binary hardening features, versions of cryptographic libraries that may lead to know CVEs, multiple CVEs for many libraries, any plaintext credentials such as private keys, weak passwords, and last but not least potential cryptographic misuses in binary level using static code analysis. The tool's open-source nature and ability to perform large-scale analysis make it a valuable resource for assessing the security of IoT/embedded devices where individuals can implement and add their own modules to enhance the tool's capabilities and address specific security analysis requirements.

The main goal of `EBAT` is to identify cryptographic misuses in binary code. One of the key contributions of `EBAT` is defining a set of cryptographic misuse rules. We have defined a total of 18 cryptographic misuse rules for various cryptographic primitives, including Symmetric Key Cryptography, Public Key Cryptography, Pseudo Random Number Generators (PRNGs), Key Derivation Functions (KDFs) and Password Based Encryption (PBE), Message Authentication Codes (MACs), Authenticated encryption/decryption and AEAD. `EBAT` implements static taint analysis (backward tracking) on the binary level using Ghidra's [70] headless scripts and various interconnected modules. For the 18 cryptographic misuse rules applied in 10 open-source cryptographic libraries with well-defined APIs, we have created rules applied in over 700 functions and 1600 functions arguments. By applying static taint analysis to these functions and arguments, we can identify violations of the cryptographic rules. Using various modules described in previous Chapters, `EBAT` is also capable of discovering cryptographic primitives and violations in wrapper functions, where it automatically updates the rules of functions to improve the detection of misuses. Overall, `EBAT's` static taint analysis provides a powerful framework for detecting the possibility of cryptographic misuse in binary code, making it a valuable tool for identifying and addressing security vulnerabilities in cryptographic implementations.

In conclusion, `EBAT` serves as a valuable resource for researchers working on firmware security. Its automated analysis process, comprehensive modules, and ability to discover possible vulnerabilities, cryptographic misuses at a binary level, and other security weaknesses make it a powerful tool for identifying and mitigating security risks in IoT/embedded devices.

## 6.2. Limitations and Future Work

This section examines the limitations and future work for `EBAT`. Although a tremendous effort is made to provide the automatic analysis as solid as possible, improvements, expansions and bug fixes are mostly welcomed. Firstly, the unpacking process is the key step in analysing firmware images. Thus, better unpacking tools, methods and algorithms are also in the scope of our future work. The firmware images that are not unpacked can be examined individually to discover the reason behind the unsuccessful unpacking process and implement or improve the ability of our tool to unpack by providing an additional module. However, encryption of the firmware image exists where the unpacking process is inevitable without acquiring the private key. `EBAT` analyses automatically the *ELF* binaries. Although other formats exist and are found by our tool, such as *PE*, we decided to improve the handling of other executable formats in the future. When it comes to *CVE* identification, we expanded our tool to handle additional *CVEs* that not only come from cryptographic libraries but other libraries as well. Although the *CVE* scanner uses only CVE Binary Tool[51], we can also implement a version scanner with *Ghidra* capabilities headless scripts (more precise but time-consuming) beyond cryptographic libraries that will allow us to spot the libraries version with better accuracy and additionally providing the reported CVEs if exists.

`EBAT` performs static taint analysis to identify violations of cryptographic rules reported in previous chapters. As mentioned, *Ghidra SRE* [70] is only used with implemented headless scripts. As *Ghidra*, newer versions have been released at the time of writing, with many new futures, bug fixes and others. Our dataset of firmware images could also execute in the latest release, which may give us more findings that older versions have missed. Furthermore, unsupported architectures may be added, or one can create one with the language specification, SLEIGH, and binaries that we cannot execute the analysis now will be possible. The newest version of *Ghidra* will be tested in future work. As with our analysis, many improvements can be implemented. Firstly, the taint propagation should fully support the use of functions such as *memcpy()*. Secondly, more tainted functions and function arguments and new cryptographic libraries API calls must be covered.

Additionally, binaries that use a cryptographic library statically linked with a binary are not supported. Future work can identify and support this feature and discover firmware's own cryptographic implementations and cryptographic function detection on obfuscated binaries. In addition, automatic binary patching on the discovered cryptographic misuses may be possible. Lastly, a framework combining static and dynamic taint analysis using *Ghidra's* emulator and QEMU is also a possible extension in our tool and left for future work.

# A

# Appendix - Results & Findings

## A.1. Evaluation Corpus

| Architecture | Bit | Endianness | # Firmwares | Percentage |
|---|---|---|---|---|
| ARC Cores Tangent-A5[a] | 32 | LE | 4 | 0.02% |
| ARM | 32 | BE | 89 | 0.39% |
| ARM | 32 | LE | 6,262 | 27.77% |
| ARM | 64 | LE | 79 | 0.35% |
| Analog Devices Blackfin[a] | 32 | LE | 6 | 0.03% |
| Intel 80386[a] | 32 | LE | 827 | 3.67% |
| MIPS | 32 | BE | 7,370 | 32.69% |
| MIPS | 32 | LE | 3,703 | 16.42% |
| MIPS | 64 | BE | 489 | 2.17% |
| Motorola m68k | 32 | BE | 2 | 0.01% |
| PowerPC | 32 | BE | 555 | 2.46% |
| Tilera TILE-Gx[a] | 32 | LE | 15 | 0.07% |
| Tilera TILE-Gx[a] | 64 | LE | 102 | 0.45% |
| Ubicom[a] | 32 | BE | 14 | 0.06% |
| No architecture[b] | 32 | LE | 5 | 0.02% |
| x86-64 | 64 | LE | 3,026 | 13.42% |

Table A.1: Various CPU Architectures over our entire dataset.

[a] Ghidra SRE[70] release version `9.1.2 (02/2020)` does not support these architectures for code analysis.

[b] Cannot find any binary architecture from the ELF header (corrupted). Possibly, the firmware image was not unpacked successfully.

| Year | # unpacked |
|---|---|
| 2002 | 1 |
| 2004 | 13 |
| 2005 | 36 |
| 2006 | 61 |
| 2007 | 93 |
| 2008 | 92 |
| 2009 | 145 |
| 2010 | 244 |
| 2011 | 322 |
| 2012 | 1,101 |
| 2013 | 1,741 |
| 2014 | 1,740 |
| 2015 | 1,755 |
| 2016 | 1,932 |
| 2017 | 3,991 |
| 2018 | 3,022 |
| 2019 | 3,227 |
| 2020 | 2,776 |
| 2021 | 256 |
| **Total** | 22,548 |

Table A.2: Successfully unpacked firmware images per year.

Table A.3: Different CPU Architectures per vendor, including unpacked and partially unpacked firmware images.

| # | Vendors | ARM | | | | MIPS | | | | PowerPC | | | | Others CPU Arch[1] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 32 bit | | 64 bit | | 32 bit | | 64 bit | | 32 bit | | 64 bit | | 32 bit | | 64 bit | |
| | | LE | BE | LE | BE | LE | BE | LE | BE | LE | BE | LE | BE | LE | BE | LE | BE |
| 1 | ASUS | 507 | 0 | 0 | 0 | 478 | 323 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | AVM | 10 | 0 | 0 | 0 | 15 | 59 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | Actiontec | 0 | 0 | 0 | 0 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 0 | 0 | 0 | 0 | 3 | 68 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | Arris | 4 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | Belkin | 3 | 1 | 0 | 0 | 20 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | Buffalo | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | D-Link | 789 | 16 | 0 | 0 | 478 | 616 | 0 | 64 | 0 | 146 | 0 | 0 | 1 | 16 | 8 | 0 |
| 10 | Dell | 45 | 0 | 0 | 0 | 0 | 4 | 0 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 35 | 0 |
| 11 | DrayTek | 4 | 0 | 0 | 0 | 84 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | EdiMax | 61 | 1 | 0 | 0 | 98 | 126 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 0 |
| 13 | FOSCAM | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | HP | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | Inmarsat | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | LinkSys | 80 | 0 | 1 | 0 | 81 | 31 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | MicroTik | 122 | 0 | 0 | 0 | 106 | 235 | 0 | 0 | 0 | 117 | 0 | 0 | 132 | 0 | 102 | 0 |
| 18 | NETGEAR | 2,320 | 58 | 1 | 0 | 802 | 1,160 | 0 | 33 | 0 | 172 | 0 | 0 | 662 | 0 | 2,853 | 0 |
| 19 | Netis | 2 | 0 | 0 | 0 | 19 | 93 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | Planet | 119 | 9 | 0 | 0 | 100 | 154 | 0 | 0 | 0 | 3 | 0 | 0 | 31 | 0 | 2 | 0 |
| 21 | QNAP | 35 | 0 | 17 | 0 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 2 | 0 |
| 22 | Rotek | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | Synology | 65 | 2 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 112 | 0 | 0 | 2 | 0 | 126 | 0 |
| 24 | TP-Link | 608 | 0 | 5 | 0 | 502 | 871 | 0 | 80 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | Tenda | 106 | 0 | 0 | 0 | 183 | 74 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| 26 | Tenvis | 3 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | Thuraya | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 0 | 0 | 2 | 0 | 84 | 58 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | Trendnet | 72 | 1 | 0 | 0 | 70 | 120 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| 30 | Ubiquiti | 948 | 0 | 36 | 0 | 163 | 2441 | 0 | 149 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 31 | Western-Digital | 0 | 0 | 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 32 | Xiaomi | 83 | 0 | 0 | 0 | 228 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 33 | Zyxel | 245 | 1 | 1 | 0 | 129 | 814 | 0 | 123 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| - | Total | 6,262 | 89 | 79 | 0 | 3,703 | 7,370 | 0 | 489 | 0 | 555 | 0 | 0 | 857 | 16 | 3,128 | 0 |

Table A.4: Overall evaluation dataset.

| # | Vendor | # Prod. | # Firm. | # unique Firmwares | # unpack Products[2] | # partially unpack[3] | # unpack [4] | Total | Earliest Firmware date | Latest Firmware date |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 265 | 1,515 | 1,468 (96.90%) | 220 (83.02%) | 42 | 1,267 | 1,309 (86.40%) | 26/01/2005 | 26/01/2021 |
| 2 | AVM | 68 | 102 | 102 (100.00%) | 68 (100.00%) | 0 | 102 | 102 (100.00%) | 02/02/2006 | 17/09/2020 |
| 3 | Actiontec | 5 | 5 | 5 (100.00%) | 5 (100.00%) | 0 | 5 | 5 (100.00%) | 19/08/2011 | 08/04/2018 |
| 4 | Addvaluetech | 4 | 4 | 4 (100.00%) | 0 (0.00%) | 0 | 0 | 0 (0.00%) | 01/06/2018 | 01/03/2021 |
| 5 | Alfa | 39 | 79 | 67 (84.81%) | 35 (89.74%) | 0 | 71 | 71 (89.87%) | 03/01/2019 | 15/12/2020 |
| 6 | Arris | 6 | 8 | 8 (100.00%) | 5 (83.33%) | 0 | 7 | 7 (87.50%) | 21/12/2015 | 29/07/2019 |
| 7 | Belkin | 28 | 64 | 64 (100.00%) | 19 (67.86%) | 2 | 43 | 45 (70.31%) | 10/01/2005 | 30/07/2018 |
| 8 | Buffalo | 12 | 15 | 12 (80.00%) | 4 (33.33%) | 0 | 4 | 4 (26.67%) | 25/04/2016 | 27/01/2021 |
| 9 | D-Link | 789 | 3,861 | 3,333 (86.32%) | 359 (45.50%) | 62 | 2,054 | 2,116 (54.80%) | 29/02/2000 | 26/04/2021 |
| 10 | Dell | 61 | 338 | 300 (88.76%) | 25 (40.98%) | 5 | 117 | 122 (36.09%) | 13/10/2011 | 16/10/2020 |
| 11 | DrayTek | 129 | 1,300 | 1,296 (99.69%) | 33 (25.58%) | 0 | 178 | 178 (13.69%) | 11/03/2004 | 23/09/2020 |
| 12 | EdiMax | 232 | 414 | 384 (92.75%) | 168 (72.41%) | 0 | 297 | 297 (71.74%) | 01/12/2005 | 29/12/2020 |
| 13 | FOSCAM | 100 | 271 | 103 (38.01%) | 4 (4.00%) | 1 | 4 | 5 (1.85%) | 04/06/2013 | 08/12/2020 |
| 14 | HP | 12 | 43 | 43 (100.00%) | 4 (33.33%) | 0 | 17 | 17 (39.53%) | 04/02/2016 | 23/06/2020 |
| 15 | Inmarsat | 35 | 44 | 44 (100.00%) | 9 (25.71%) | 0 | 11 | 11 (25.00%) | 17/12/2008 | 01/10/2020 |
| 16 | LinkSys | 140 | 250 | 207 (82.80%) | 109 (77.86%) | 0 | 195 | 195 (78.00%) | 14/12/2011 | 05/02/2021 |
| 17 | MicroTik | 20 | 826 | 826 (100.00%) | 9 (45.00%) | 0 | 814 | 814 (98.55%) | 07/07/2011 | 09/02/2021 |
| 18 | NETGEAR | 829 | 9,458 | 3,790 (40.07%) | 553 (66.71%) | 194 | 7,867 | 8,061 (85.23%) | 31/08/2001 | 07/04/2021 |
| 19 | Netis | 42 | 129 | 124 (96.12%) | 35 (83.33%) | 0 | 114 | 114 (88.37%) | 25/01/2012 | 11/11/2020 |
| 20 | Planet | 290 | 816 | 718 (87.99%) | 162 (55.86%) | 13 | 405 | 418 (51.23%) | 23/06/2003 | 20/02/2021 |
| 21 | QNAP | 238 | 4,566 | 2,526 (55.32%) | 10 (4.20%) | 65 | 44 | 109 (2.39%) | 28/10/2014 | 28/04/2021 |
| 22 | Rotek | 1 | 1 | 1 (100.00%) | 1 (100.00%) | 0 | 1 | 1 (100.00%) | 02/06/2020 | 02/06/2020 |
| 23 | Synology | 61 | 319 | 318 (99.69%) | 61 (100.00%) | 0 | 319 | 319 (100.00%) | 21/10/2014 | 03/09/2020 |
| 24 | TP-Link | 950 | 3,258 | 3,210 (98.53%) | 640 (67.37%) | 11 | 2,058 | 2,069 (63.51%) | 14/03/2006 | 02/04/2021 |
| 25 | Tenda | 267 | 707 | 699 (98.87%) | 135 (50.56%) | 4 | 363 | 367 (51.91%) | 04/11/2009 | 31/03/2021 |
| 26 | Tenvis | 3 | 13 | 13 (100.00%) | 3 (100.00%) | 1 | 6 | 7 (53.85%) | 07/11/2012 | 04/07/2014 |
| 27 | Thuraya | 7 | 22 | 22 (100.00%) | 1 (14.29%) | 0 | 2 | 2 (9.09%) | 05/07/2011 | 01/05/2020 |
| 28 | Totolink | 65 | 157 | 151 (96.18%) | 57 (87.69%) | 0 | 144 | 144 (91.72%) | 02/06/2015 | 23/03/2021 |
| 29 | Trendnet | 370 | 548 | 450 (82.12%) | 182 (49.19%) | 3 | 264 | 267 (48.72%) | 13/01/2003 | 01/03/2021 |
| 30 | Ubiquiti | 253 | 3,773 | 935 (24.78%) | 249 (98.42%) | 0 | 3,737 | 3,737 (99.05%) | 07/11/2012 | 10/02/2021 |
| 31 | Western-Digital | 2 | 5 | 5 (100.00%) | 2 (100.00%) | 0 | 5 | 5 (100.00%) | 14/12/2012 | 28/05/2020 |
| 32 | Xiaomi | 15 | 313 | 313 (100.00%) | 15 (100.00%) | 1 | 312 | 313 (100.00%) | 29/07/2014 | 22/07/2020 |
| 33 | Zyxel | 515 | 2,849 | 2,825 (99.16%) | 231 (44.85%) | 20 | 1,297 | 1317 (46.23%) | 22/04/1997 | 24/03/2021 |
| - | **Total** | 5,853 | 36,073 | 24,366 (67.55%) | 3,413 (58.31%) | 424 | 22,124 | 22,548 (62.51%) | - | - |

[2] Unpack Products have one or more unpack firmware images, including the partial ones.
[3] A firmware image is marked as partially unpacked if at least one binary is found, but some required dynamic libraries are not present.
[4] A firmware image is marked as successfully unpacked if at least one binary is found during the extraction process and all required dynamic libraries are present.

## A.2. Binary Statistics

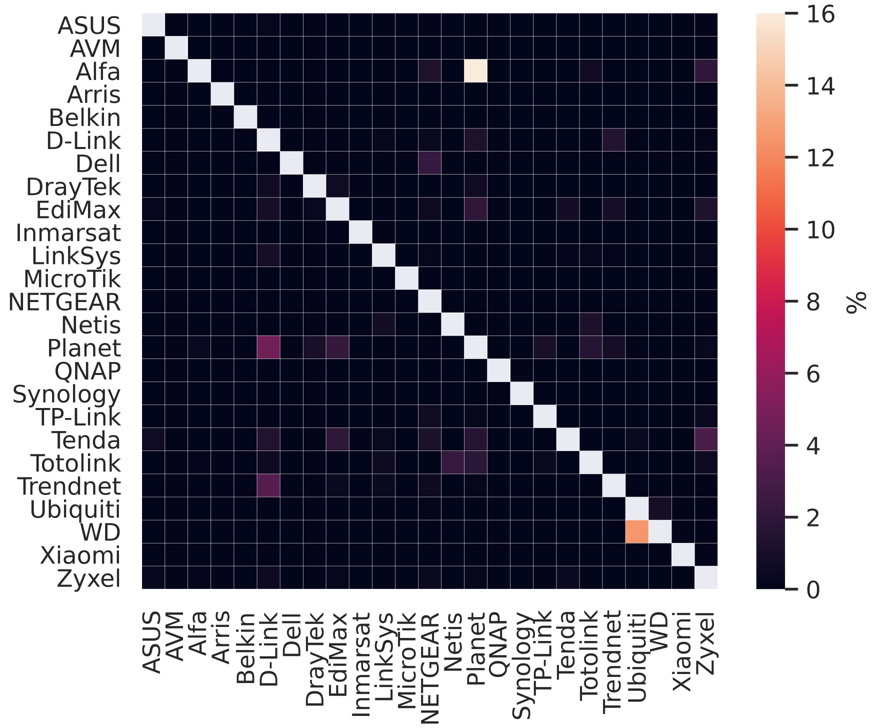



Figure A.1: Heatmap of executable binaries that use a cryptographic library.

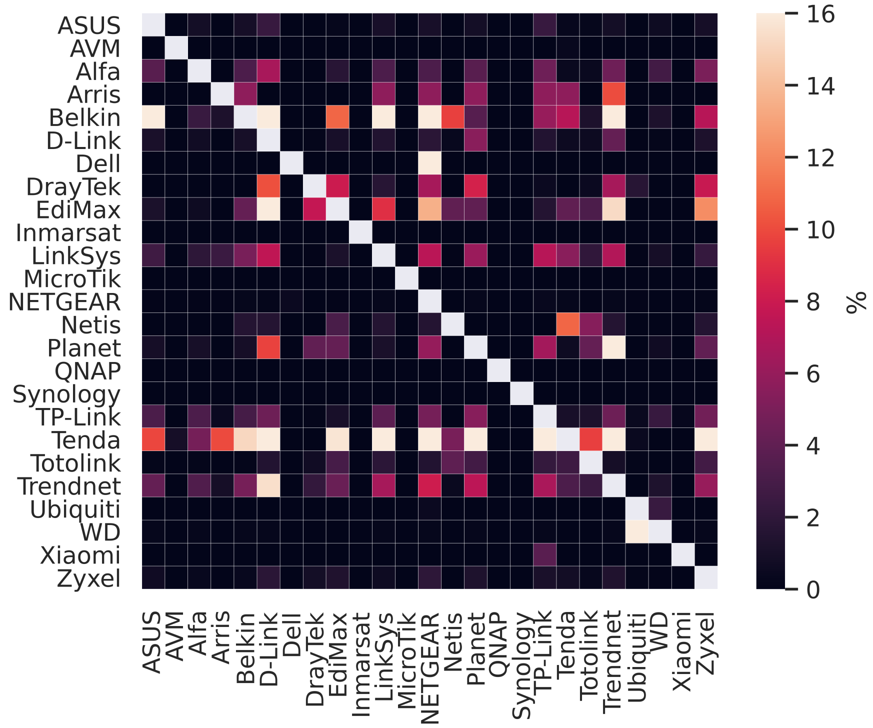



Figure A.2: Heatmap of library binaries that use a cryptographic library.

Table A.5: Overall binaries statistics per vendors' firmware images.

| # | Vendor | # binaries | # libraries | % | # executables | % | # unique binaries | % | # unique libs | % | # unique exec | % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 456,405 | 321,015 | 70.34 | 135,390 | 29.66 | 40,651 | 8.91 | 23,722 | 7.39 | 16,931 | 12.51 |
| 2 | AVM | 6,702 | 3,942 | 58.82 | 2,760 | 41.18 | 3,601 | 53.73 | 2,217 | 56.24 | 1,384 | 50.14 |
| 3 | Actiontec | 859 | 430 | 50.06 | 429 | 49.94 | 853 | 99.30 | 426 | 99.07 | 427 | 99.53 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 13,975 | 9,373 | 67.07 | 4,602 | 32.93 | 2,458 | 17.59 | 1,643 | 17.53 | 815 | 17.71 |
| 6 | Arris | 2,867 | 2,054 | 71.64 | 813 | 28.36 | 1,501 | 52.35 | 1,085 | 52.82 | 416 | 51.17 |
| 7 | Belkin | 5,351 | 2,836 | 53.00 | 2,515 | 47.00 | 3,077 | 57.50 | 1,363 | 48.06 | 1,714 | 68.15 |
| 8 | Buffalo | 754 | 471 | 62.47 | 283 | 37.53 | 520 | 68.97 | 305 | 64.76 | 215 | 75.97 |
| 9 | D-Link | 411,767 | 203,247 | 49.36 | 208,520 | 50.64 | 71,015 | 17.25 | 32,367 | 15.92 | 38,648 | 18.53 |
| 10 | Dell | 20,384 | 15,471 | 75.90 | 4,913 | 24.10 | 1,928 | 9.46 | 1,387 | 8.97 | 541 | 11.01 |
| 11 | DrayTek | 20,565 | 11,330 | 55.09 | 9,235 | 44.91 | 5,245 | 25.50 | 2,509 | 22.14 | 2,739 | 29.66 |
| 12 | EdiMax | 22,919 | 11,407 | 49.77 | 11,512 | 50.23 | 8,164 | 35.62 | 3,516 | 30.82 | 4,649 | 40.38 |
| 13 | FOSCAM | 362 | 250 | 69.06 | 112 | 30.94 | 271 | 74.86 | 182 | 72.80 | 89 | 79.46 |
| 14 | HP | 498 | 187 | 37.55 | 311 | 62.45 | 81 | 16.27 | 33 | 17.65 | 48 | 15.43 |
| 15 | Inmarsat | 3,723 | 2,174 | 58.39 | 1,549 | 41.61 | 2,674 | 71.82 | 1,555 | 71.53 | 1,119 | 72.24 |
| 16 | LinkSys | 64,442 | 38,049 | 59.04 | 26,393 | 40.96 | 19,873 | 30.84 | 11,588 | 30.46 | 8,285 | 31.39 |
| 17 | MicroTik | 425,468 | 325,851 | 76.59 | 99,617 | 23.41 | 238,791 | 56.12 | 217,926 | 66.88 | 20,865 | 20.95 |
| 18 | NETGEAR | 8,991,403 | 5,110,754 | 56.84 | 3,880,649 | 43.16 | 168,274 | 1.87 | 95,549 | 1.87 | 72,729 | 1.87 |
| 19 | Netis | 8,572 | 2,409 | 28.10 | 6,163 | 71.90 | 2,182 | 25.45 | 692 | 28.73 | 1,490 | 24.18 |
| 20 | Planet | 70,238 | 44,631 | 63.54 | 25,607 | 36.46 | 19,713 | 28.07 | 11,375 | 25.49 | 8,340 | 32.57 |
| 21 | QNAP | 47,239 | 30,115 | 63.75 | 17,124 | 36.25 | 8,461 | 17.91 | 4,643 | 15.42 | 3,818 | 22.30 |
| 22 | Rotek | 177 | 116 | 65.54 | 61 | 34.46 | 177 | 100.00 | 116 | 100.00 | 61 | 100.00 |
| 23 | Synology | 680,704 | 437,764 | 64.31 | 242,940 | 35.69 | 94,806 | 13.93 | 59,107 | 13.50 | 35,699 | 14.69 |
| 24 | TP-Link | 450,346 | 312,598 | 69.41 | 137,748 | 30.59 | 97,042 | 21.55 | 59,368 | 18.99 | 37,674 | 27.35 |
| 25 | Tenda | 46,662 | 28,491 | 61.06 | 18,171 | 38.94 | 11,194 | 23.99 | 6,179 | 21.69 | 5,015 | 27.60 |
| 26 | Tenvis | 363 | 137 | 37.74 | 226 | 62.26 | 243 | 66.94 | 93 | 67.88 | 150 | 66.37 |
| 27 | Thuraya | 346 | 292 | 84.39 | 54 | 15.61 | 173 | 50.00 | 146 | 50.00 | 27 | 50.00 |
| 28 | Totolink | 16,420 | 8,109 | 49.38 | 8,311 | 50.62 | 7,795 | 47.47 | 4,012 | 49.48 | 3,783 | 45.52 |
| 29 | Trendnet | 31,914 | 19,228 | 60.25 | 12,686 | 39.75 | 13,938 | 43.67 | 7,583 | 39.44 | 6,356 | 50.10 |
| 30 | Ubiquiti | 1,345,814 | 999,938 | 74.30 | 345,876 | 25.70 | 59,474 | 4.42 | 42,037 | 4.20 | 17,437 | 5.04 |
| 31 | Western-Digital | 12,805 | 7,716 | 60.26 | 5,089 | 39.74 | 5,858 | 45.75 | 3,544 | 45.93 | 2,314 | 45.47 |
| 32 | Xiaomi | 126,331 | 78,055 | 61.79 | 48,276 | 38.21 | 13,877 | 10.98 | 6,712 | 8.60 | 7,165 | 14.84 |
| 33 | Zyxel | 415,538 | 238,640 | 57.43 | 176,898 | 42.57 | 85,219 | 20.51 | 48,687 | 20.40 | 36,533 | 20.65 |
| - | **Total** | 13,701,913 | 8,267,080 | 60.34 | 5,434,833 | 39.66 | 989,129 | 7.22 | 651,667 | 7.88 | 337,476 | 6.21 |

Table A.6: Overall analyzed binaries statistics across vendors.

| # | Vendor | binaries | # analysed binaries | % | libraries | # analysed libraries | % | executables | # analysed executables | % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 456,405 | 59,484 | 6.22 | 321,015 | 28,076 | 8.75 | 135,390 | 31,408 | 23.20 |
| 2 | AVM | 6,702 | 612 | 3.71 | 3,942 | 245 | 6.22 | 2,760 | 367 | 13.30 |
| 3 | Actiontec | 859 | 198 | 0.81 | 430 | 5 | 1.16 | 429 | 193 | 44.99 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0 | 0.00 | 0 | 0 | 0.00 |
| 5 | Alfa | 13,975 | 518 | 1.52 | 9,373 | 209 | 2.23 | 4,602 | 309 | 6.71 |
| 6 | Arris | 2,867 | 309 | 1.80 | 2,054 | 49 | 2.39 | 813 | 260 | 31.98 |
| 7 | Belkin | 5,351 | 117 | 0.47 | 2,836 | 24 | 0.85 | 2,515 | 93 | 3.70 |
| 8 | Buffalo | 754 | 26 | 0.82 | 471 | 6 | 1.27 | 283 | 20 | 7.07 |
| 9 | D-Link | 411,767 | 59,426 | 5.60 | 203,247 | 22,688 | 11.16 | 208,520 | 36,738 | 17.62 |
| 10 | Dell | 20,384 | 3,851 | 12.42 | 15,471 | 2,519 | 16.28 | 4,913 | 1,332 | 27.11 |
| 11 | DrayTek | 20,565 | 3,816 | 2.86 | 11,330 | 556 | 4.91 | 9,235 | 3,260 | 35.30 |
| 12 | EdiMax | 22,919 | 1,271 | 0.43 | 11,407 | 87 | 0.76 | 11,512 | 1,184 | 10.28 |
| 13 | FOSCAM | 362 | 17 | 0.87 | 250 | 3 | 1.20 | 112 | 14 | 12.50 |
| 14 | HP | 498 | 22 | 0.04 | 187 | 0 | 0.00 | 311 | 22 | 7.07 |
| 15 | Inmarsat | 3,723 | 298 | 1.91 | 2,174 | 69 | 3.17 | 1,549 | 229 | 14.78 |
| 16 | LinkSys | 64,442 | 3,649 | 1.30 | 38,049 | 807 | 2.12 | 26,393 | 2,842 | 10.77 |
| 17 | MicroTik | 425,468 | 9,374 | 0.15 | 325,851 | 539 | 0.17 | 99,617 | 8,835 | 8.87 |
| 18 | NETGEAR | 8,991,403 | 859,333 | 4.02 | 5,110,754 | 356,226 | 6.97 | 3,880,649 | 503,107 | 12.96 |
| 19 | Netis | 8,572 | 268 | 0.09 | 2,409 | 5 | 0.21 | 6,163 | 263 | 4.27 |
| 20 | Planet | 70,238 | 6,575 | 2.16 | 44,631 | 1,464 | 3.28 | 25,607 | 5,111 | 19.96 |
| 21 | QNAP | 47,239 | 9,821 | 7.69 | 30,115 | 3,570 | 11.85 | 17,124 | 6,251 | 36.50 |
| 22 | Rotek | 177 | 5 | 0.03 | 116 | 0 | 0.00 | 61 | 5 | 8.20 |
| 23 | Synology | 680,704 | 228,040 | 16.76 | 437,764 | 112,921 | 25.79 | 242,940 | 115,119 | 47.39 |
| 24 | TP-Link | 450,346 | 27,163 | 1.66 | 312,598 | 7,297 | 2.33 | 137,748 | 19,866 | 14.42 |
| 25 | Tenda | 46,662 | 2,379 | 0.39 | 28,491 | 160 | 0.56 | 18,171 | 2,219 | 12.21 |
| 26 | Tenvis | 363 | 129 | 2.54 | 137 | 8 | 5.84 | 226 | 121 | 53.54 |
| 27 | Thuraya | 346 | 10 | 0.60 | 292 | 2 | 0.68 | 54 | 8 | 14.81 |
| 28 | Totolink | 16,420 | 1,029 | 1.52 | 8,109 | 241 | 2.97 | 8,311 | 788 | 9.48 |
| 29 | Trendnet | 31,914 | 2,386 | 1.67 | 19,228 | 513 | 2.67 | 12,686 | 1,873 | 14.76 |
| 30 | Ubiquiti | 1,345,814 | 94,850 | 2.32 | 999,938 | 30,526 | 3.05 | 345,876 | 64,324 | 18.60 |
| 31 | Western-Digital | 12,805 | 1,349 | 2.88 | 7,716 | 359 | 4.65 | 5,089 | 990 | 19.45 |
| 32 | Xiaomi | 126,331 | 16,059 | 3.21 | 78,055 | 3,932 | 5.04 | 48,276 | 12,127 | 25.12 |
| 33 | Zyxel | 415,538 | 59,655 | 4.19 | 238,640 | 16,987 | 7.12 | 176,898 | 42,668 | 24.12 |
| - | **Total** | 13,701,913 | 1,452,039 | 4.37 | 8,267,080 | 590,093 | 7.14 | 5,434,833 | 861,946 | 15.86 |

## A.3. Firmware Update

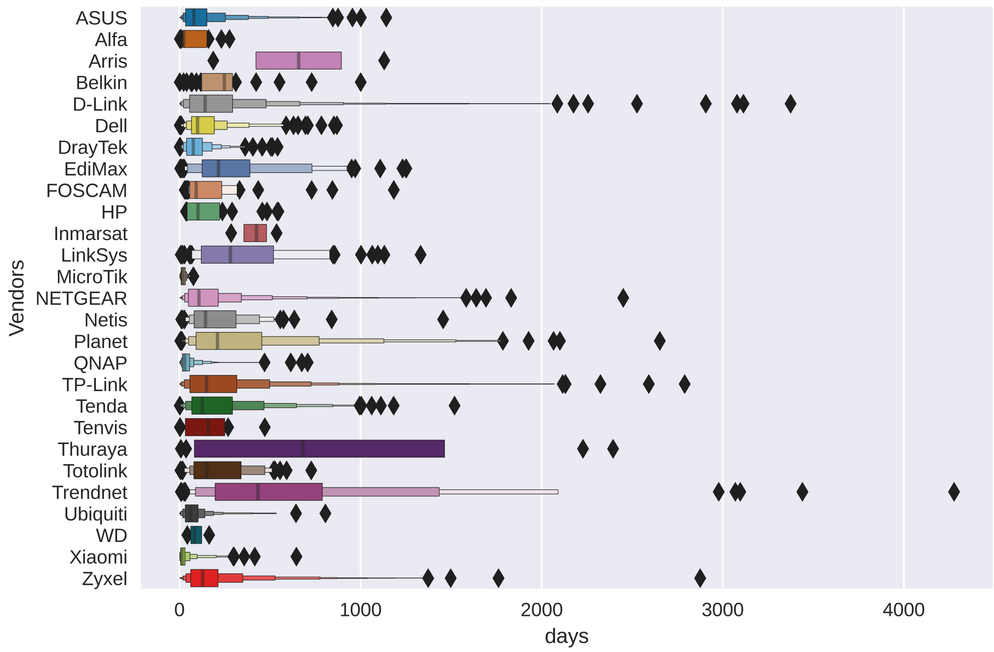



Figure A.3: Firmware update gap in days over all vendors.

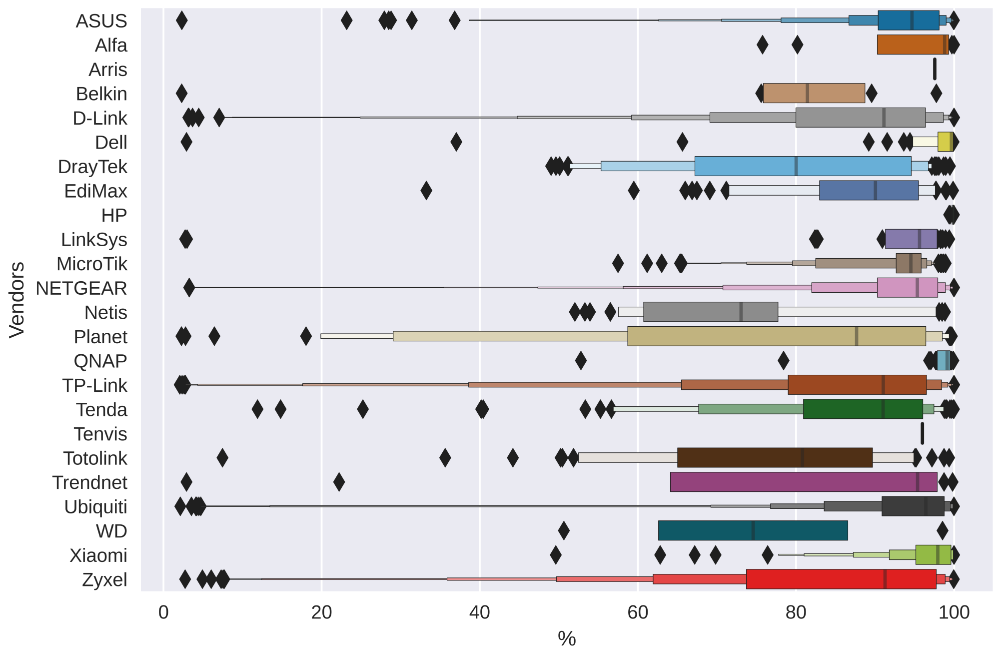



Figure A.4: Firmware update over all binaries.

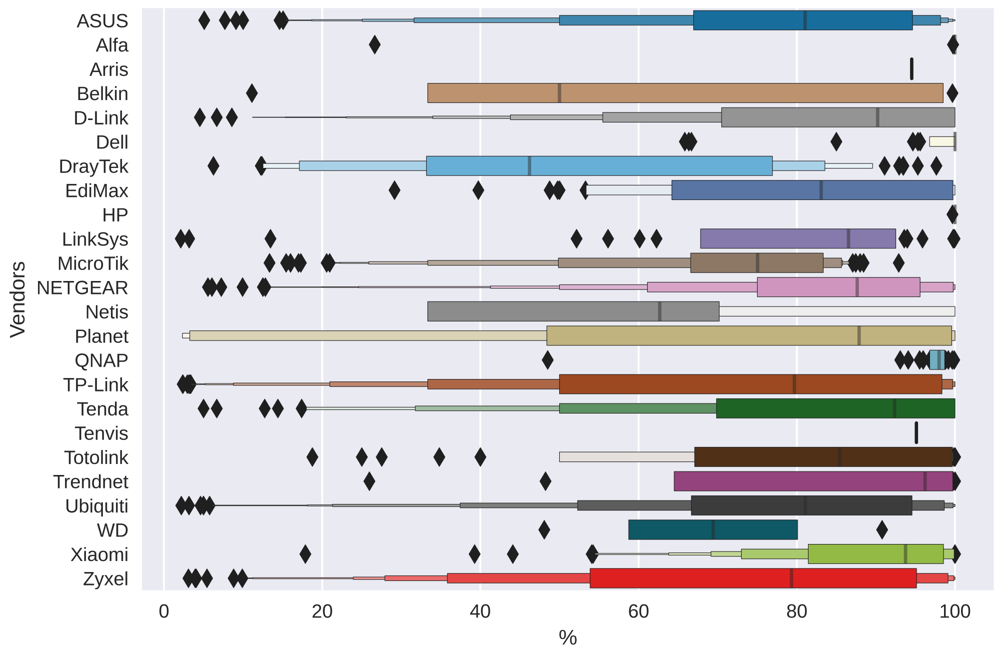



Figure A.5: Firmware updates only binaries that use a cryptographic library (executables and libraries).

| Mean values | | | | | |
|---|---|---|---|---|---|
| # | Vendors | days | # | Vendors | days |
| 1 | ASUS | 125.75 | 2 | AVM | 402.93 |
| 5 | Alfa | 79.00 | 6 | Arris | 658.00 |
| 7 | Belkin | 261.68 | 9 | D-Link | 241.10 |
| 10 | Dell | 147.51 | 11 | DrayTek | 94.86 |
| 12 | EdiMax | 325.80 | 13 | FOSCAM | 223.66 |
| 14 | HP | 176.21 | 15 | Inmarsat | 415.33 |
| 16 | LinkSys | 395.56 | 17 | MicroTik | 20.76 |
| 18 | NETGEAR | 118.26 | 19 | Netis | 209.13 |
| 20 | Planet | 366.32 | 21 | QNAP | 34.95 |
| 23 | Synology | 140.72 | 24 | TP-Link | 240.02 |
| 25 | Tenda | 215.46 | 26 | Tenvis | 176.67 |
| 27 | Thuraya | 882.00 | 28 | Totolink | 210.45 |
| 29 | Trendnet | 667.98 | 30 | Ubiquiti | 69.54 |
| 31 | WD | 96.00 | 32 | Xiaomi | 32.47 |
| 33 | Zyxel | 184.12 | - | - | - |

Table A.7: Firmware update gap mean values in days over all vendors.

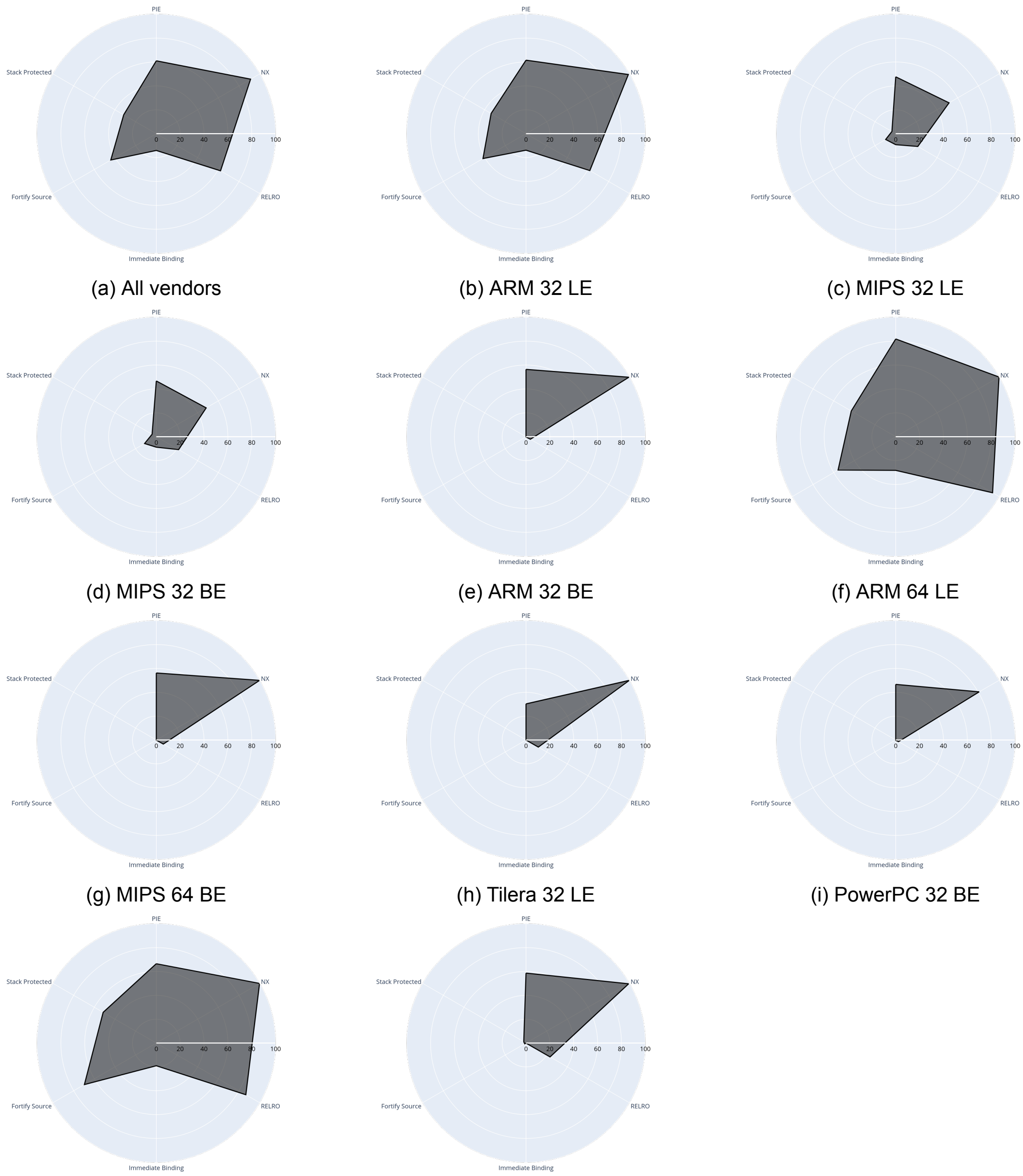
| Mean values | | |
|---|---|---|
| # | Vendors | % |
| 1 | ASUS | 92.27 |
| 3 | Alfa | 93.91 |
| 4 | Arris | 97.56 |
| 5 | Belkin | 71.35 |
| 6 | D-Link | 85.91 |
| 7 | Dell | 95.08 |
| 8 | DrayTek | 78.40 |
| 9 | EdiMax | 87.38 |
| 10 | HP | 99.79 |
| 12 | LinkSys | 87.73 |
| 13 | MicroTik | 92.26 |
| 14 | NETGEAR | 92.38 |
| 15 | Netis | 73.15 |
| 16 | Planet | 77.08 |
| 17 | QNAP | 95.94 |
| 18 | Synology | 89.08 |
| 19 | TP-Link | 83.24 |
| 20 | Tenda | 84.75 |
| 21 | Tenvis | 95.99 |
| 22 | Totolink | 75.60 |
| 23 | Trendnet | 75.40 |
| 24 | Ubiquiti | 91.01 |
| 25 | WD | 74.58 |
| 26 | Xiaomi | 96.08 |
| 27 | Zyxel | 82.91 |

Table A.8: Mean values of the percentage of firmware update similarity calculated for all binaries.

| Mean values | | |
|---|---|---|
| # | Vendors | % |
| 1 | ASUS | 76.18 |
| 2 | Alfa | 96.61 |
| 3 | Arris | 94.53 |
| 4 | Belkin | 58.52 |
| 5 | D-Link | 82.93 |
| 6 | Dell | 97.12 |
| 7 | DrayTek | 52.35 |
| 8 | EdiMax | 79.03 |
| 9 | HP | 99.93 |
| 10 | LinkSys | 77.12 |
| 11 | MicroTik | 70.61 |
| 12 | NETGEAR | 84.85 |
| 13 | Netis | 58.75 |
| 14 | Planet | 72.30 |
| 15 | QNAP | 95.52 |
| 16 | Synology | 79.21 |
| 17 | TP-Link | 71.07 |
| 18 | Tenda | 81.22 |
| 19 | Tenvis | 95.11 |
| 20 | Totolink | 78.86 |
| 21 | Trendnet | 78.66 |
| 22 | Ubiquiti | 76.59 |
| 23 | WD | 69.42 |
| 24 | Xiaomi | 88.77 |
| 25 | Zyxel | 71.88 |

Table A.9: Mean values of the percentage of firmware update similarity calculated for ‘crypto’ binaries.

# A.4. Exploit mitigation techniques on firmware images




(a) All vendors
(b) ARM 32 LE
(c) MIPS 32 LE
(d) MIPS 32 BE
(e) ARM 32 BE
(f) ARM 64 LE
(g) MIPS 64 BE
(h) Tilera 32 LE
(i) PowerPC 32 BE
(j) Intel x86-64 LE
(k) Intel-80386 32 LE

Figure A.6: Exploit mitigation techniques on firmware images by CPU architecture.

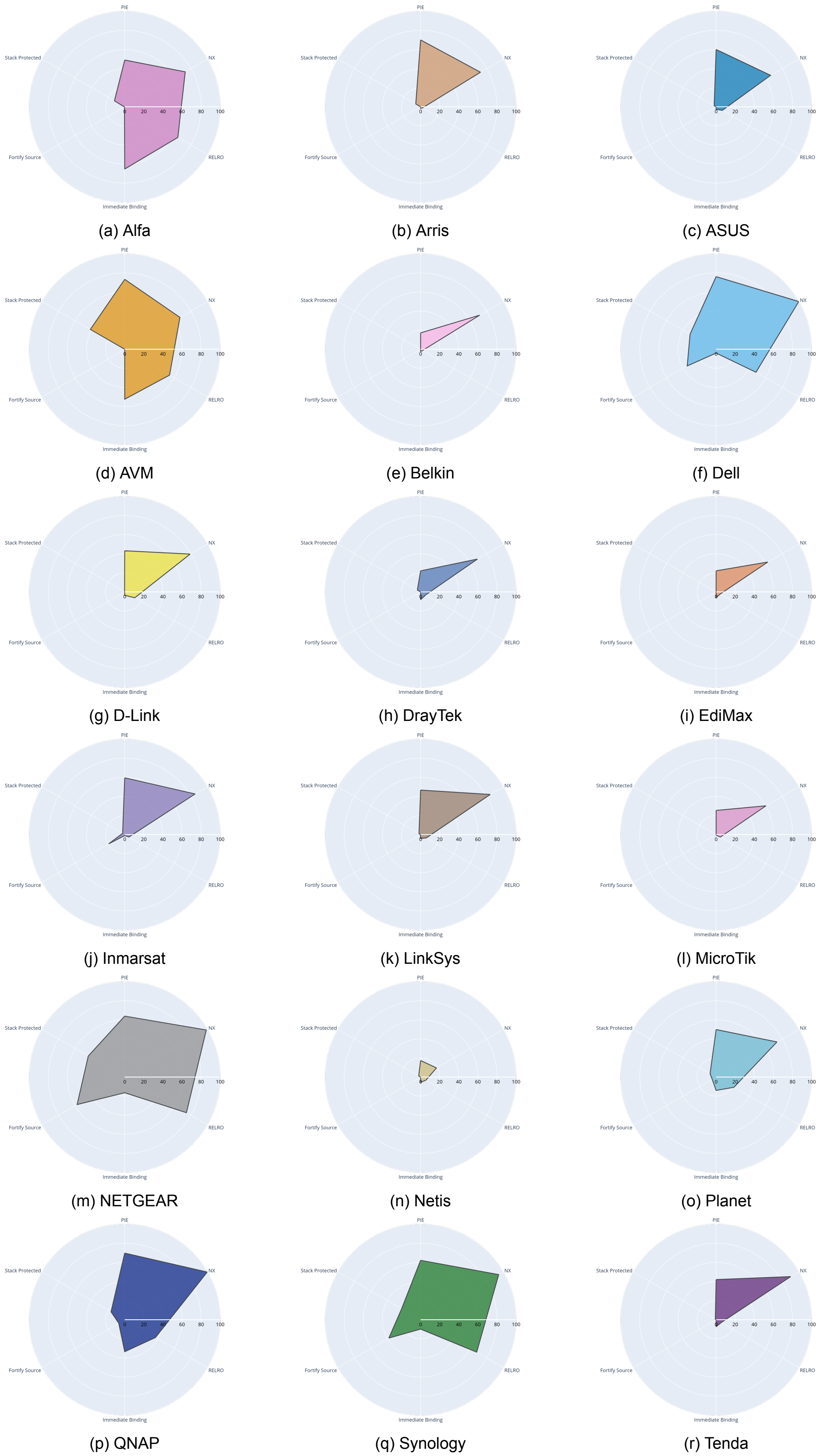
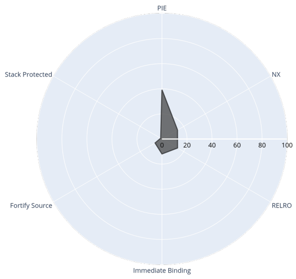
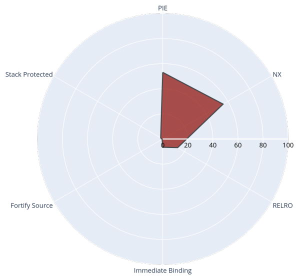


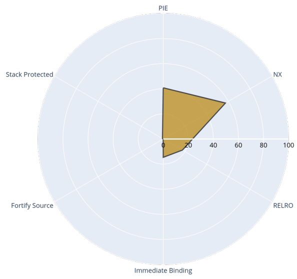

Figure A.7: Exploit mitigation techniques on firmware images by Vendor.



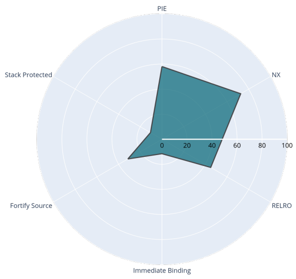

(a) Totolink



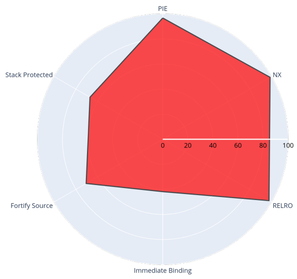

(b) TP-Link



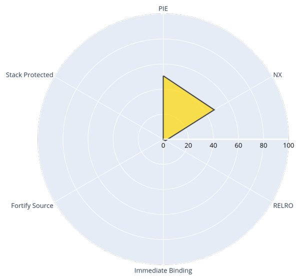

(c) Trendnet



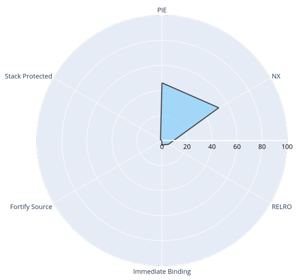

(d) Ubiquiti



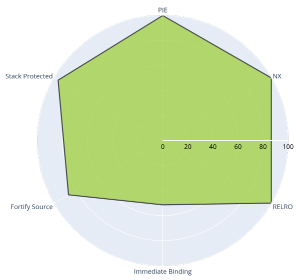

(e) Western Digital




(f) Xiaomi




(g) Zyxel




(h) Ubuntu Server

Figure A.8: Exploit mitigation techniques on firmware images by Vendor (continued).

Table A.10: Overall exploit mitigation techniques across vendors (PIE and NX bit).

| # | Vendor | # binaries | PIE | | | NX bit | | |
|---|---|---|---|---|---|---|---|---|
| | | | ✓ | ✗ | nf | ✓ | ✗ | nf |
| 1 | ASUS | 456,405 | 204,165 (44.73%) | 136,021 (29.80%) | 116,219 (25.46%) | 224,180 (49.12%) | 116,006 (25.42%) | 116,219 (25.46%) |
| 2 | AVM | 6,702 | 3,875 (57.82%) | 1,411 (21.05%) | 1,416 (21.13%) | 3,530 (52.67%) | 1,756 (26.20%) | 1,416 (21.13%) |
| 3 | Actiontec | 859 | 207 (24.10%) | 441 (51.34%) | 211 (24.56%) | 648 (75.44%) | 0 (0.00%) | 211 (24.56%) |
| 4 | Addvaluetech | 0 | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) |
| 5 | Alfa | 13,975 | 4,517 (32.32%) | 4,708 (33.69%) | 4,750 (33.99%) | 6,755 (48.34%) | 2,470 (17.67%) | 4,750 (33.99%) |
| 6 | Arris | 2,867 | 1,536 (53.58%) | 657 (22.92%) | 674 (23.51%) | 1,588 (55.39%) | 605 (21.10%) | 674 (23.51%) |
| 7 | Belkin | 5,351 | 811 (15.16%) | 3,915 (73.16%) | 625 (11.68%) | 3,376 (63.09%) | 1,350 (25.23%) | 625 (11.68%) |
| 8 | Buffalo | 754 | 157 (20.82%) | 299 (39.66%) | 298 (39.52%) | 339 (44.96%) | 117 (15.52%) | 298 (39.52%) |
| 9 | D-Link | 411,767 | 161,267 (39.16%) | 213,317 (51.81%) | 37,183 (9.03%) | 296,255 (71.95%) | 78,334 (19.02%) | 37,178 (9.03%) |
| 10 | Dell | 20,384 | 14,141 (69.37%) | 4,440 (21.78%) | 1,803 (8.85%) | 18,581 (91.15%) | 0 (0.00%) | 1,803 (8.85%) |
| 11 | DrayTek | 20,565 | 3,604 (17.52%) | 12,747 (61.98%) | 4,214 (20.49%) | 11,231 (54.61%) | 5,119 (24.89%) | 4,215 (20.50%) |
| 12 | EdiMax | 22,919 | 3891 (16.98%) | 13,733 (59.92%) | 5,295 (23.10%) | 11,008 (48.03%) | 6,616 (28.87%) | 5,295 (23.10%) |
| 13 | FOSCAM | 362 | 139 (38.40%) | 107 (29.56%) | 116 (32.04%) | 120 (33.15%) | 126 (34.81%) | 116 (32.04%) |
| 14 | HP | 498 | 153 (30.72%) | 311 (62.45%) | 34 (6.83%) | 464 (93.17%) | 0 (0.00%) | 34 (6.83%) |
| 15 | Inmarsat | 3,723 | 2,205 (59.23%) | 1,498 (40.24%) | 20 (0.54%) | 3,153 (84.69%) | 550 (14.77%) | 20 (0.54%) |
| 16 | LinkSys | 64,442 | 26,452 (41.05%) | 30,276 (46.98%) | 7,714 (11.97%) | 47,734 (74.07%) | 8,996 (13.96%) | 7,712 (11.97%) |
| 17 | MicroTik | 425,468 | 31,240 (7.34%) | 92,017 (21.63%) | 302,211 (71.03%) | 74,385 (17.48%) | 48,872 (11.49%) | 302,211 (71.03%) |
| 18 | NETGEAR | 8,991,403 | 5,610,305 (62.40%) | 3,171,164 (35.27%) | 209,934 (2.33%) | 8,664,145 (96.36%) | 117,329 (1.30%) | 209,929 (2.33%) |
| 19 | Netis | 8,572 | 1,455 (16.97%) | 6,953 (81.11%) | 164 (1.91%) | 1,613 (18.82%) | 6,798 (79.30%) | 161 (1.88%) |
| 20 | Planet | 70,238 | 26,007 (37.03%) | 26,230 (37.34%) | 18,001 (25.63%) | 38,426 (54.71%) | 13,811 (19.66%) | 18,001 (25.63%) |
| 21 | QNAP | 47,239 | 31,149 (65.94%) | 13,553 (28.69%) | 2,537 (5.37%) | 44,620 (94.46%) | 82 (0.17%) | 2,537 (5.37%) |
| 22 | Rotek | 177 | 116 (65.54%) | 61 (34.46%) | 0 (0.00%) | 1 (0.56%) | 176 (99.44%) | 0 (0.00%) |
| 23 | Synology | 680,704 | 367,275 (53.96%) | 223,669 (32.86%) | 89,760 (13.19%) | 558,203 (82.00%) | 32,741 (4.81%) | 89,760 (13.19%) |
| 24 | TP-Link | 450,346 | 159,347 (35.38%) | 140,959 (31.30%) | 150,040 (33.32%) | 166,907 (37.06%) | 133,398 (29.62%) | 150,041 (33.32%) |
| 25 | Tenda | 46,662 | 16,042 (34.38%) | 22,132 (47.43%) | 8,488 (18.19%) | 34,430 (73.79%) | 3,744 (8.02%) | 8,488 (18.19%) |
| 26 | Tenvis | 363 | 44 (12.12%) | 260 (71.63%) | 59 (16.25%) | 292 (80.44%) | 12 (3.31%) | 59 (16.25%) |
| 27 | Thuraya | 346 | 96 (27.75%) | 54 (15.61%) | 196 (56.65%) | 12 (3.47%) | 138 (39.88%) | 196 (56.65%) |
| 28 | Totolink | 16,420 | 5,361 (32.65%) | 8,324 (50.69%) | 2,735 (16.66%) | 1,960 (11.94%) | 11,725 (71.41%) | 2,735 (16.66%) |
| 29 | Trendnet | 31,914 | 9,842 (30.84%) | 14,312 (44.85%) | 7,760 (24.32%) | 13,848 (43.39%) | 10,324 (32.35%) | 7,742 (24.26%) |
| 30 | Ubiquiti | 1,345,814 | 526,014 (39.09%) | 380,625 (28.28%) | 439,175 (32.63%) | 658,610 (48.94%) | 248,059 (18.43%) | 439,145 (32.63%) |
| 31 | Western-Digital | 12,805 | 12,315 (96.17%) | 416 (3.25%) | 74 (0.58%) | 12,586 (98.29%) | 145 (1.13%) | 74 (0.58%) |
| 32 | Xiaomi | 126,331 | 49,483 (39.17%) | 48,274 (38.21%) | 28,574 (22.62%) | 45,900 (36.33%) | 51,857 (41.05%) | 28,574 (22.62%) |
| 33 | Zyxel | 415,538 | 157,978 (38.02%) | 185,028 (44.53%) | 72,532 (17.45%) | 179,655 (43.23%) | 163,351 (39.31%) | 72,532 (17.45%) |
| - | **Total** | 13,701,913 | 7,431,189 (54.23%) | 4,757,912 (34.72%) | 1,512,812 (11.04%) | 11,124,555 (81.19%) | 1,064,607 (7.77%) | 1,512,751 (11.04%) |

Table A.11: Overall exploit mitigation techniques across vendors (Stack Protected and Fortify Source).

| # | Vendor | # binaries | Stack Protected | | | Fortify Source | | |
|---|---|---|---|---|---|---|---|---|
| | | | ✓ | ✗ | nf | ✓ | ✗ | nf |
| 1 | ASUS | 456,405 | 10,809 (2.37%) | 445,596 (97.63%) | 0 (0.00%) | 115 (0.03%) | 373,847 (81.91%) | 82,443 (18.06%) |
| 2 | AVM | 6,702 | 2,787 (41.58%) | 3,915 (58.42%) | 0 (0.00%) | 0 (0.00%) | 5,446 (81.26%) | 1,256 (18.74%) |
| 3 | Actiontec | 859 | 0 (0.00%) | 859 (100.00%) | 0 (0.00%) | 0 (0.00%) | 193 (22.47%) | 666 (77.53%) |
| 4 | Addvaluetech | 0 | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) |
| 5 | Alfa | 13,975 | 1,738 (12.44%) | 12,237 (87.56%) | 0 (0.00%) | 13 (0.09%) | 4,364 (31.23%) | 9,598 (68.68%) |
| 6 | Arris | 2,867 | 168 (5.86%) | 2,699 (94.14%) | 0 (0.00%) | 0 (0.00%) | 2,469 (86.12%) | 398 (13.88%) |
| 7 | Belkin | 5,351 | 0 (0.00%) | 5,351 (100.00%) | 0 (0.00%) | 0 (0.00%) | 4,574 (85.48%) | 777 (14.52%) |
| 8 | Buffalo | 754 | 98 (13.00%) | 656 (87.00%) | 0 (0.00%) | 0 (0.00%) | 234 (31.03%) | 520 (68.97%) |
| 9 | D-Link | 411,767 | 629 (0.15%) | 411,133 (99.85%) | 5 (0.00%) | 158 (0.04%) | 353,319 (85.81%) | 58,290 (14.16%) |
| 10 | Dell | 20,384 | 6,428 (31.53%) | 13,956 (68.47%) | 0 (0.00%) | 6,209 (30.46%) | 11,547 (56.65%) | 2,628 (12.89%) |
| 11 | DrayTek | 20,565 | 813 (3.95%) | 19,751 (96.04%) | 1 (0.00%) | 0 (0.00%) | 15,984 (77.72%) | 4,581 (22.28%) |
| 12 | EdiMax | 22,919 | 9 (0.04%) | 22,910 (99.96%) | 0 (0.00%) | 10 (0.04%) | 16,720 (72.95%) | 6,189 (27.00%) |
| 13 | FOSCAM | 362 | 15 (4.14%) | 347 (95.86%) | 0 (0.00%) | 0 (0.00%) | 274 (75.69%) | 88 (24.31%) |
| 14 | HP | 498 | 0 (0.00%) | 498 (100.00%) | 0 (0.00%) | 0 (0.00%) | 434 (87.15%) | 64 (12.85%) |
| 15 | Inmarsat | 3,723 | 95 (2.55%) | 3,628 (97.45%) | 0 (0.00%) | 659 (17.70%) | 2,726 (73.22%) | 338 (9.08%) |
| 16 | LinkSys | 64,442 | 1.254 (1.95%) | 63,186 (98.05%) | 2 (0.00%) | 147 (0.23%) | 53,035 (82.30%) | 11,260 (17.47%) |
| 17 | MicroTik | 425,468 | 0 (0.00%) | 425,468 (100.00%) | 0 (0.00%) | 0 (0.00%) | 253,381 (59.55%) | 172,087 (40.45%) |
| 18 | NETGEAR | 8,991,403 | 3,963,228 (44.08%) | 5,028,169 (55.92%) | 6 (0.00%) | 3,854,109 (42.86%) | 2,839,895 (31.58%) | 2,297,399 (25.55%) |
| 19 | Netis | 8,572 | 200 (2.33%) | 8,369 (97.63%) | 3 (0.03%) | 0 (0.00%) | 4,963 (57.90%) | 3,609 (42.10%) |
| 20 | Planet | 70,238 | 5,145 (7.33%) | 65,093 (92.67%) | 0 (0.00%) | 2,264 (3.22%) | 45,354 (64.57%) | 22,620 (32.20%) |
| 21 | QNAP | 47,239 | 7,827 (16.57%) | 39,412 (83.43%) | 0 (0.00%) | 2,755 (5.83%) | 35,220 (74.56%) | 9,264 (19.61%) |
| 22 | Rotek | 177 | 0 (0.00%) | 177 (100.00%) | 0 (0.00%) | 0 (0.00%) | 1 (0.56%) | 176 (99.44%) |
| 23 | Synology | 680,704 | 156,260 (22.96%) | 524,444 (77.04%) | 0 (0.00%) | 153,127 (22.50%) | 245,495 (36.06%) | 282,082 (41.44%) |
| 24 | TP-Link | 450,346 | 8,510 (1.89%) | 441,813 (98.11%) | 23 (0.01%) | 1,446 (0.32%) | 263,666 (58.55%) | 185,234 (41.13%) |
| 25 | Tenda | 46,662 | 456 (0.98%) | 46,206 (99.02%) | 0 (0.00%) | 0 (0.00%) | 37,547 (80.47%) | 9,115 (19.53%) |
| 26 | Tenvis | 363 | 0 (0.00%) | 363 (100.00%) | 0 (0.00%) | 0 (0.00%) | 317 (87.33%) | 46 (12.67%) |
| 27 | Thuraya | 346 | 0 (0.00%) | 346 (100.00%) | 0 (0.00%) | 0 (0.00%) | 94 (27.17%) | 252 (72.83%) |
| 28 | Totolink | 16,420 | 189 (1.15%) | 16,231 (98.85%) | 0 (0.00%) | 337 (2.05%) | 4,984 (30.35%) | 11,099 (67.59%) |
| 29 | Trendnet | 31,914 | 226 (0.71%) | 31,670 (99.24%) | 18 (0.06%) | 11 (0.03%) | 19,880 (62.29%) | 12,023 (37.67%) |
| 30 | Ubiquiti | 1,345,814 | 142,581 (10.59%) | 1,203,203 (89.40%) | 30 (0.00%) | 190,137 (14.13%) | 414,104 (30.77%) | 741,573 (55.10%) |
| 31 | Ubuntu | 722 | 696 (96.40%) | 26 (3.60%) | 0 (0.00%) | 388 (53.74%) | 59 (8.17%) | 275 (38.09%) |
| 32 | Western-Digital | 12,805 | 8,581 (67.01%) | 4,224 (32.99%) | 0 (0.00%) | 6,526 (50.96%) | 2,727 (21.30%) | 3,552 (27.74%) |
| 33 | Xiaomi | 126,331 | 0 (0.00%) | 126,331 (100.00%) | 0 (0.00%) | 0 (0.00%) | 57,788 (45.74%) | 68,543 (54.26%) |
| 34 | Zyxel | 415,538 | 5,672 (1.36%) | 409,866 (98.64%) | 0 (0.00%) | 609 (0.15%) | 285,485 (68.70%) | 129,444 (31.15%) |
| - | **Total** | 13,701,913 | 4,323,718 (31.56%) | 9,378,107 (68.44%) | 88 (0.00%) | 4,218,632 (30.79%) | 5,356,067 (39.09%) | 4,127,214 (30.12%) |

Table A.12: Overall exploit mitigation techniques across vendors (RELRO and Immediate binding).

| # | Vendor | # binaries | RELRO | | | Immediate binding | | |
|---|---|---|---|---|---|---|---|---|
| | | | ✓ | ✗ | nf | ✓ | ✗ | nf |
| 1 | ASUS | 456,405 | 25,536 (5.60%) | 314,650 (68.94%) | 116,219 (25.46%) | 8,868 (1.94%) | 331,318 (72.59%) | 116,219 (25.46%) |
| 2 | AVM | 6,702 | 2,867 (42.78%) | 2,419 (36.09%) | 1416 (21.13%) | 2,767 (41.29%) | 2,519 (37.59%) | 1,416 (21.13%) |
| 3 | Actiontec | 859 | 12 (1.40%) | 636 (74.04%) | 211 (24.56%) | 20 (2.33%) | 628 (73.11%) | 211 (24.56%) |
| 4 | Addvaluetech | 0 | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) | 0 (0.00%) |
| 5 | Alfa | 13,975 | 5,928 (42.42%) | 3,297 (23.59%) | 4,750 (33.99%) | 6,017 (43.06%) | 3,208 (22.96%) | 4,750 (33.99%) |
| 6 | Arris | 2,867 | 51 (1.78%) | 2,142 (74.71%) | 674 (23.51%) | 35 (1.22%) | 2,158 (75.27%) | 674 (23.51%) |
| 7 | Belkin | 5,351 | 116 (2.17%) | 4,610 (86.15%) | 625 (11.68%) | 116 (2.17%) | 4,610 (86.15%) | 625 (11.68%) |
| 8 | Buffalo | 754 | 240 (31.83%) | 216 (28.65%) | 298 (39.52%) | 242 (32.10%) | 214 (28.38%) | 298 (39.52%) |
| 9 | D-Link | 411,767 | 45,538 (11.06%) | 329,046 (79.91%) | 37,183 (9.03%) | 13,093 (3.18%) | 361,491 (87.79%) | 37,183 (9.03%) |
| 10 | Dell | 20,384 | 8,953 (43.92%) | 9,628 (47.23%) | 1,803 (8.85%) | 746 (3.66%) | 17,835 (87.50%) | 1,803 (8.85%) |
| 11 | DrayTek | 20,565 | 1,010 (4.91%) | 15,341 (74.60%) | 4,214 (20.49%) | 1,331 (6.47%) | 15,020 (73.04%) | 4,214 (20.49%) |
| 12 | EdiMax | 22,919 | 831 (3.63%) | 16,793 (73.27%) | 5,295 (23.10%) | 1,040 (4.54%) | 16,584 (72.36%) | 5,295 (23.10%) |
| 13 | FOSCAM | 362 | 5 (1.38%) | 241 (66.57%) | 116 (32.04%) | 0 (0.00%) | 246 (67.96%) | 116 (32.04%) |
| 14 | HP | 498 | 136 (27.31%) | 328 (65.86%) | 34 (6.83%) | 136 (27.31%) | 328 (65.86%) | 34 (6.83%) |
| 15 | Inmarsat | 3,723 | 189 (5.08%) | 3,514 (94.39%) | 20 (0.54%) | 39 (1.05%) | 3,664 (98.42%) | 20 (0.54%) |
| 16 | LinkSys | 64,442 | 3,953 (6.13%) | 52,775 (81.90%) | 7,714 (11.97%) | 2,319 (3.60%) | 54,409 (84.43%) | 7,714 (11.97%) |
| 17 | MicroTik | 425,468 | 6,323 (1.49%) | 116,934 (27.48%) | 302,211 (71.03%) | 697 (0.16%) | 122,560 (28.81%) | 302,211 (71.03%) |
| 18 | NETGEAR | 8,991,403 | 6,555,149 (72.90%) | 2,226,320 (24.76%) | 209,934 (2.33%) | 1,443,002 (16.05%) | 7,338,467 (81.62%) | 209,934 (2.33%) |
| 19 | Netis | 8,572 | 544 (6.35%) | 7,864 (91.74%) | 164 (1.91%) | 464 (5.41%) | 7,944 (92.67%) | 164 (1.91%) |
| 20 | Planet | 70,238 | 11,216 (15.97%) | 41,021 (58.40%) | 18,001 (25.63%) | 7,300 (10.39%) | 44,937 (63.98%) | 18,001 (25.63%) |
| 21 | QNAP | 47,239 | 16,626 (35.20%) | 28,076 (59.43%) | 2,537 (5.37%) | 14,960 (31.67%) | 29,742 (62.96%) | 2,537 (5.37%) |
| 22 | Rotek | 177 | 1 (0.56%) | 176 (99.44%) | 0 (0.00%) | 0 (0.00%) | 177 (100.00%) | 0 (0.00%) |
| 23 | Synology | 680,704 | 400,914 (58.90%) | 190,030 (27.92%) | 89,760 (13.19%) | 57,932 (8.51%) | 533,012 (78.30%) | 89,760 (13.19%) |
| 24 | TP-Link | 450,346 | 41,262 (9.16%) | 259,044 (57.52%) | 150,040 (33.32%) | 19,812 (4.40%) | 280,494 (62.28%) | 150,040 (33.32%) |
| 25 | Tenda | 46,662 | 2,311 (4.95%) | 35,863 (76.86%) | 8,488 (18.19%) | 2,801 (6.00%) | 35,373 (75.81%) | 8,488 (18.19%) |
| 26 | Tenvis | 363 | 12 (3.31%) | 292 (80.44%) | 59 (16.25%) | 38 (10.47%) | 266 (73.28%) | 59 (16.25%) |
| 27 | Thuraya | 346 | 6 (1.73%) | 144 (41.62%) | 196 (56.65%) | 2 (0.58%) | 148 (42.77%) | 196 (56.65%) |
| 28 | Totolink | 16,420 | 1,962 (11.95%) | 11,723 (71.39%) | 2,735 (16.66%) | 1,623 (9.88%) | 12,062 (73.46%) | 2,735 (16.66%) |
| 29 | Trendnet | 31,914 | 4,269 (13.38%) | 19,885 (62.31%) | 7,760 (24.32%) | 3,523 (11.04%) | 20,631 (64.65%) | 7,760 (24.32%) |
| 30 | Ubiquiti | 1,345,814 | 407,754 (30.30%) | 498,885 (37.07%) | 439,175 (32.63%) | 104,403 (7.76%) | 802,236 (59.61%) | 439,175 (32.63%) |
| 31 | Ubuntu | 722 | 722 (100.00%) | 0 (0.00%) | 0 (0.00%) | 371 (51.39%) | 351 (48.61%) | 0 (0.00%) |
| 32 | Western-Digital | 12,805 | 12,461 (97.31%) | 270 (2.11%) | 74 (0.58%) | 5,319 (41.54%) | 7,412 (57.88%) | 74 (0.58%) |
| 33 | Xiaomi | 126,331 | 2,120 (1.68%) | 95,637 (75.70%) | 28,574 (22.62%) | 313 (0.25%) | 97,444 (77.13%) | 28,574 (22.62%) |
| 34 | Zyxel | 415,538 | 20,669 (4.97%) | 322,337 (77.57%) | 72,532 (17.45%) | 12,249 (2.95%) | 330,757 (79.60%) | 72,532 (17.45%) |
| - | **Total** | 13,701,913 | 7,578,964 (55.31%) | 4,610,137 (33.65%) | 1,512,812 (11.04%) | 1,711,207 (12.49%) | 10,477,894 (76.47%) | 1,512,812 (11.04%) |

# A.5. Credentials and Password hashes

Table A.13: Overall Credentials statistics by vendor.

| # | Vendor | SSH Private Key | | SSH Private Key (encrypted) | | SSH Public Key | | PGP Signatures | | PKCS12 (encrypted) | | PKCS12 (decrypted) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | # | # Firm. | # | # Firm. | # | # Firm. | # | # Firm. | # | # Firm. | # | # Firm. |
| 1 | ASUS | 66 | 41 | 0 | 0 | 66 | 41 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | AVM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | Actiontec | 11 | 4 | 1 | 1 | 11 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | Arris | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | Belkin | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | Buffalo | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | D-Link | 597 | 306 | 105 | 105 | 597 | 306 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | Dell | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | DrayTek | 2 | 1 | 0 | 0 | 2 | 1 | 0 | 0 | 40 | 40 | 135 | 45 |
| 12 | EdiMax | 16 | 8 | 1 | 1 | 16 | 8 | 0 | 0 | 6 | 6 | 18 | 6 |
| 13 | FOSCAM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | HP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | Inmarsat | 6 | 3 | 2 | 2 | 6 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | LinkSys | 204 | 92 | 6 | 3 | 204 | 92 | 20 | 4 | 0 | 0 | 0 | 0 |
| 17 | MicroTik | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | NETGEAR | 291 | 122 | 3,904 | 106 | 291 | 122 | 36,367 | 141 | 0 | 0 | 0 | 0 |
| 19 | Netis | 3 | 3 | 0 | 0 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | Planet | 87 | 35 | 2 | 2 | 87 | 35 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | QNAP | 48 | 24 | 4 | 4 | 48 | 24 | 0 | 0 | 10 | 10 | 23 | 23 |
| 22 | Rotek | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | Synology | 0 | 0 | 0 | 0 | 0 | 0 | 484 | 163 | 0 | 0 | 0 | 0 |
| 24 | TP-Link | 10 | 10 | 0 | 0 | 10 | 10 | 1 | 1 | 1 | 1 | 0 | 0 |
| 25 | Tenda | 46 | 40 | 0 | 0 | 46 | 40 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | Tenvis | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | Thuraya | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 0 | 0 | 0 | 0 | 11 | 11 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | Trendnet | 65 | 21 | 0 | 0 | 65 | 21 | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | Ubiquiti | 13 | 1 | 1 | 1 | 22 | 1 | 3043 | 209 | 0 | 0 | 224 | 130 |
| 31 | Western-Digital | 24 | 4 | 3 | 3 | 24 | 4 | 70 | 4 | 0 | 0 | 0 | 0 |
| 32 | Xiaomi | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 33 | Zyxel | 363 | 260 | 10 | 10 | 363 | 260 | 0 | 0 | 33 | 33 | 292 | 116 |
| - | **Total** | 1,852 | 975 | 4,041 | 240 | 1,872 | 986 | 39,986 | 523 | 90 | 90 | 692 | 320 |

Table A.14: Overall Password Hashes.

| # | Vendors | # firmwares | # hashes | # cracked | # public cracked | # unique | # unique cracked | # public unique cracked | Hash Types: DES (Unix) ✓ | DES (Unix) ✗ | MD5 (Unix) ✓ | MD5 (Unix) ✗ | MD5 (ARP) ✓ | MD5 (ARP) ✗ | Blowfish (Unix) ✓ | Blowfish (Unix) ✗ | SHA256 (Unix) ✓ | SHA256 (Unix) ✗ | SHA512 (Unix) ✓ | SHA512 (Unix) ✗ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 179 | 179 | 179 | 177 | 5 | 5 | 4 | 4 | 0 | 175 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | AVM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | Actiontec | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 58 | 70 | 67 | 22 | 9 | 7 | 3 | 0 | 0 | 67 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | Arris | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | Belkin | 4 | 7 | 7 | 7 | 3 | 3 | 3 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | Buffalo | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | D-Link | 429 | 959 | 750 | 743 | 53 | 40 | 38 | 327 | 45 | 291 | 164 | 132 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | Dell | 64 | 64 | 64 | 22 | 2 | 2 | 1 | 42 | 0 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | DrayTek | 31 | 33 | 30 | 30 | 6 | 4 | 4 | 5 | 0 | 25 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | EdiMax | 88 | 114 | 107 | 107 | 23 | 18 | 18 | 6 | 0 | 101 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | FOSCAM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | HP | 17 | 17 | 17 | 0 | 1 | 1 | 0 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | Inmarsat | 9 | 28 | 9 | 9 | 13 | 5 | 5 | 0 | 0 | 9 | 16 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| 16 | LinkSys | 16 | 17 | 12 | 11 | 9 | 5 | 4 | 0 | 1 | 12 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 17 | MicroTik | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | NETGEAR | 498 | 999 | 821 | 613 | 44 | 37 | 31 | 263 | 121 | 430 | 57 | 16 | 0 | 0 | 0 | 0 | 0 | 112 | 0 |
| 19 | Netis | 103 | 105 | 103 | 103 | 4 | 3 | 3 | 94 | 2 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | Planet | 170 | 256 | 154 | 127 | 71 | 52 | 42 | 53 | 12 | 101 | 89 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 21 | QNAP | 26 | 49 | 46 | 46 | 4 | 2 | 2 | 0 | 0 | 46 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | Rotek | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | Synology | 318 | 435 | 429 | 429 | 3 | 2 | 2 | 0 | 0 | 429 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | TP-Link | 1,777 | 1,992 | 703 | 611 | 434 | 17 | 10 | 89 | 0 | 614 | 1,289 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | Tenda | 282 | 1,397 | 1,396 | 914 | 24 | 23 | 15 | 892 | 1 | 500 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| 26 | Tenvis | 2 | 6 | 4 | 2 | 3 | 2 | 1 | 0 | 2 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | Thuraya | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 120 | 174 | 173 | 173 | 12 | 11 | 11 | 36 | 0 | 137 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | Trendnet | 75 | 98 | 87 | 79 | 27 | 24 | 22 | 45 | 0 | 41 | 8 | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| 30 | Ubiquiti | 731 | 738 | 668 | 668 | 43 | 42 | 42 | 539 | 0 | 102 | 70 | 0 | 0 | 0 | 0 | 27 | 0 | 0 | 0 |
| 31 | Western-Digital | 4 | 31 | 21 | 19 | 25 | 18 | 17 | 0 | 0 | 4 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 5 |
| 32 | Xiaomi | 312 | 521 | 222 | 222 | 2 | 1 | 1 | 0 | 0 | 222 | 299 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 33 | Zyxel | 411 | 688 | 593 | 593 | 42 | 30 | 30 | 25 | 2 | 568 | 93 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - | **Total** | 5,730 | 8,983 | 6,668 | 5,733 | 793 | 290 | 252 | 2,437 | 186 | 3,922 | 2,115 | 149 | 0 | 4 | 1 | 27 | 6 | 129 | 7 |

Table A.15: Overall Credentials statistics by vendor (continued).

| # | Vendor | Certificates | | Public Key | | Private Key | | Private Key (encrypted) | | Private Key (decrypted) | | Cryptographic Parameters | | Certificate Signing Request (CSR) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | # | # Firm. | # | # Firm. | # | # Firm. | # | # Firm. | # | # Firm. | # | # Firm. | # | # Firm. |
| 1 | ASUS | 102,648 | 867 | 1,338 | 1,011 | 199 | 155 | 191 | 167 | 12 | 12 | 716 | 711 | 34 | 17 |
| 2 | AVM | 409 | 15 | 0 | 0 | 0 | 0 | 11 | 11 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | Actiontec | 22 | 5 | 7 | 5 | 7 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 2,760 | 30 | 31 | 19 | 29 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | Arris | 15 | 7 | 8 | 4 | 8 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | Belkin | 28 | 10 | 25 | 16 | 25 | 10 | 4 | 4 | 0 | 0 | 6 | 3 | 0 | 0 |
| 8 | Buffalo | 5 | 2 | 5 | 2 | 4 | 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 9 | D-Link | 63,041 | 1,187 | 3,074 | 1,336 | 3,074 | 1,358 | 87 | 79 | 0 | 0 | 133 | 101 | 260 | 129 |
| 10 | Dell | 81 | 9 | 81 | 9 | 81 | 9 | 27 | 9 | 0 | 0 | 27 | 9 | 27 | 9 |
| 11 | DrayTek | 521 | 175 | 565 | 174 | 310 | 175 | 42 | 42 | 255 | 85 | 75 | 75 | 1 | 1 |
| 12 | EdiMax | 4,952 | 114 | 160 | 25 | 136 | 106 | 36 | 36 | 24 | 12 | 1 | 1 | 11 | 7 |
| 13 | FOSCAM | 160 | 4 | 3 | 3 | 3 | 3 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 14 | HP | 0 | 0 | 17 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | Inmarsat | 14 | 2 | 6 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 16 | LinkSys | 2,726 | 137 | 240 | 126 | 185 | 134 | 13 | 13 | 0 | 0 | 15 | 12 | 0 | 0 |
| 17 | MicroTik | 0 | 0 | 70 | 70 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | NETGEAR | 676,190 | 1,573 | 6,933 | 1,369 | 3,057 | 1,295 | 220 | 161 | 0 | 0 | 670 | 533 | 268 | 132 |
| 19 | Netis | 24 | 8 | 6 | 5 | 6 | 6 | 7 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | Planet | 1,315 | 184 | 313 | 155 | 313 | 168 | 41 | 41 | 0 | 0 | 4 | 4 | 2 | 1 |
| 21 | QNAP | 5,788 | 32 | 210 | 25 | 158 | 26 | 9 | 9 | 46 | 23 | 0 | 0 | 23 | 23 |
| 22 | Rotek | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | Synology | 46,929 | 261 | 792 | 307 | 147 | 88 | 22 | 22 | 0 | 0 | 465 | 155 | 0 | 0 |
| 24 | TP-Link | 3,022 | 1,245 | 1,278 | 846 | 1,267 | 894 | 53 | 53 | 0 | 0 | 29 | 29 | 0 | 0 |
| 25 | Tenda | 1,242 | 170 | 850 | 153 | 850 | 186 | 1 | 1 | 0 | 0 | 210 | 42 | 1 | 1 |
| 26 | Tenvis | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | Thuraya | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 180 | 81 | 147 | 77 | 147 | 81 | 13 | 13 | 0 | 0 | 3 | 3 | 0 | 0 |
| 29 | Trendnet | 385 | 75 | 134 | 91 | 131 | 99 | 13 | 13 | 3 | 3 | 11 | 11 | 4 | 2 |
| 30 | Ubiquiti | 300,743 | 782 | 3,915 | 592 | 3460 | 571 | 241 | 139 | 455 | 131 | 804 | 210 | 2 | 1 |
| 31 | Western-Digital | 719 | 5 | 80 | 5 | 31 | 5 | 1 | 1 | 0 | 0 | 64 | 4 | 0 | 0 |
| 32 | Xiaomi | 31,568 | 299 | 1,379 | 313 | 90 | 84 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 33 | Zyxel | 33,453 | 701 | 1,603 | 813 | 1155 | 819 | 146 | 146 | 448 | 126 | 315 | 299 | 338 | 150 |
| - | **Total** | 1,278,943 | 7,981 | 23,271 | 7,572 | 14,877 | 6,305 | 1,182 | 970 | 1,244 | 393 | 3,549 | 2,203 | 972 | 474 |

# A.6. Cryptographic Libraries

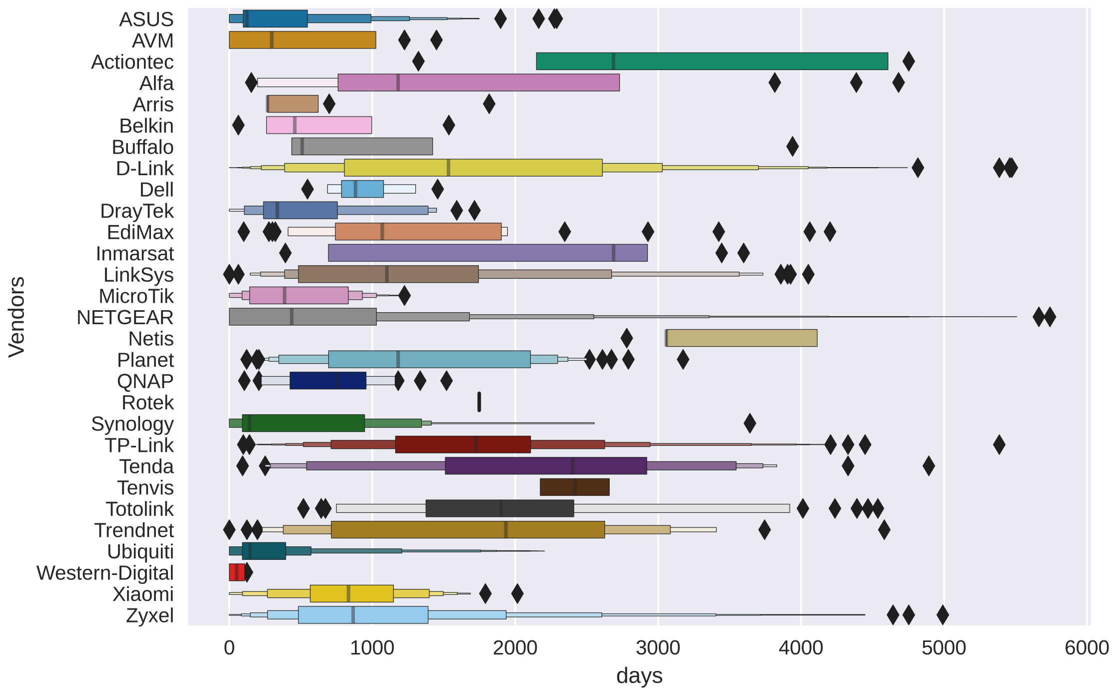



Figure A.9: OpenSSL outdated versions timegap.

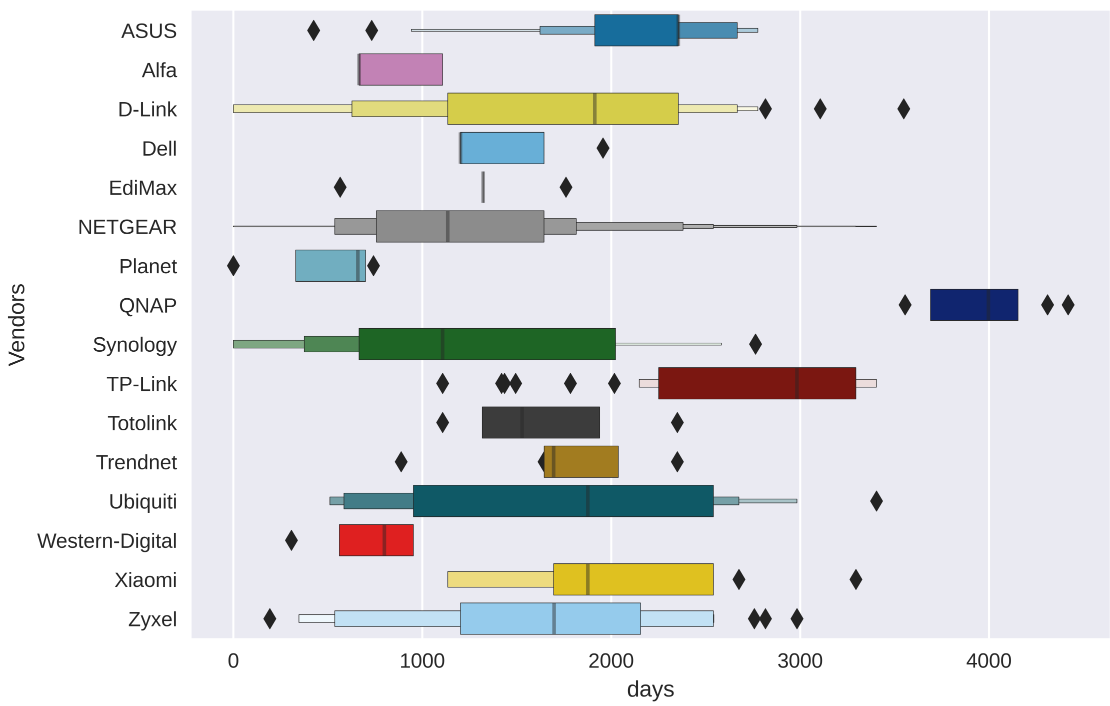



Figure A.10: GnuPG outdated versions timegap.

| # | Vendor | #days |
|---|---|---|
| 1 | ASUS | 361.75 |
| 2 | AVM | 515.19 |
| 3 | Actiontec | 3,104.20 |
| 5 | Alfa | 1,729.37 |
| 6 | Arris | 590.57 |
| 7 | Belkin | 685.33 |
| 8 | Buffalo | 1,349.25 |
| 9 | D-Link | 1,721.23 |
| 10 | Dell | 938.15 |
| 11 | DrayTek | 564.35 |
| 12 | EdiMax | 1,322.37 |
| 15 | Inmarsat | 2,003.56 |
| 16 | LinkSys | 1,343.24 |
| 17 | MicroTik | 459.45 |
| 18 | NETGEAR | 780.76 |
| 19 | Netis | 3,424.00 |
| 20 | Planet | 1,327.17 |
| 21 | QNAP | 711.76 |
| 22 | Rotek | 1,748.00 |
| 23 | Synology | 549.72 |
| 24 | TP-Link | 1,719.40 |
| 25 | Tenda | 2,263.28 |
| 26 | Tenvis | 2,417.00 |
| 28 | Totolink | 2,101.17 |
| 29 | Trendnet | 1,808.38 |
| 30 | Ubiquiti | 327.53 |
| 31 | Western-Digital | 56.25 |
| 32 | Xiaomi | 830.33 |
| 33 | Zyxel | 1,046.77 |
| - | **Mean** | 1,303.43 |

Table A.16: OpenSSL time-gap outdated versions for each vendor's firmware image mean values.

| # | Vendor | #days |
|---|---|---|
| 1 | ASUS | 2,211.38 |
| 5 | Alfa | 830.75 |
| 9 | D-Link | 1,649.56 |
| 10 | Dell | 1,425.59 |
| 12 | EdiMax | 1,258.20 |
| 18 | NETGEAR | 1,168.45 |
| 20 | Planet | 466.67 |
| 21 | QNAP | 3,965.09 |
| 23 | Synology | 1,227.20 |
| 24 | TP-Link | 2,765.98 |
| 28 | Totolink | 1,661.67 |
| 29 | Trendnet | 1,823.69 |
| 30 | Ubiquiti | 1,727.94 |
| 31 | Western-Digital | 714.75 |
| 32 | Xiaomi | 1,941.25 |
| 33 | Zyxel | 1,610.59 |
| - | **Mean** | 1,653.05 |

Table A.17: GnuPG time-gap outdated versions for each vendor's firmware image mean values.

| # | Vendor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9-12 | 13-16 | 17-20 | 21-25 | 26-max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 231 | 121 | 117 | 186 | 63 | 107 | 60 | 12 | 17 | 68 | 57 | 20 | 2 | 0 |
| 2 | AVM | 5 | 0 | 1 | 3 | 1 | 0 | 0 | 0 | 1 | 4 | 1 | 0 | 0 | 0 |
| 3 | Actiontec | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 2 |
| 5 | Alfa | 0 | 8 | 1 | 3 | 0 | 0 | 1 | 4 | 0 | 4 | 16 | 0 | 0 | 25 |
| 6 | Arris | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 0 |
| 7 | Belkin | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 8 | Buffalo | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 9 | D-Link | 10 | 17 | 47 | 42 | 62 | 49 | 44 | 30 | 36 | 218 | 188 | 222 | 190 | 264 |
| 10 | Dell | 0 | 0 | 0 | 0 | 0 | 3 | 6 | 9 | 3 | 13 | 7 | 0 | 0 | 0 |
| 11 | DrayTek | 7 | 35 | 12 | 15 | 32 | 10 | 10 | 1 | 6 | 8 | 5 | 32 | 1 | 0 |
| 12 | EdiMax | 0 | 1 | 1 | 2 | 1 | 1 | 4 | 0 | 2 | 11 | 10 | 2 | 2 | 4 |
| 15 | Inmarsat | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 5 |
| 16 | LinkSys | 2 | 2 | 4 | 6 | 16 | 9 | 4 | 14 | 2 | 23 | 23 | 19 | 11 | 31 |
| 17 | MicroTik | 72 | 60 | 67 | 116 | 64 | 77 | 68 | 80 | 63 | 147 | 0 | 0 | 0 | 0 |
| 18 | NETGEAR | 1661 | 393 | 197 | 250 | 718 | 160 | 290 | 73 | 163 | 1251 | 294 | 246 | 237 | 334 |
| 19 | Netis | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| 20 | Planet | 0 | 4 | 9 | 7 | 3 | 22 | 13 | 10 | 5 | 21 | 58 | 14 | 37 | 38 |
| 21 | QNAP | 0 | 2 | 1 | 0 | 2 | 3 | 3 | 5 | 4 | 9 | 3 | 1 | 0 | 0 |
| 22 | Rotek | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 23 | Synology | 62 | 66 | 9 | 43 | 3 | 11 | 8 | 31 | 5 | 65 | 0 | 0 | 7 | 2 |
| 24 | TP-Link | 0 | 3 | 10 | 5 | 20 | 19 | 15 | 46 | 19 | 86 | 180 | 217 | 314 | 257 |
| 25 | Tenda | 0 | 0 | 5 | 0 | 5 | 5 | 2 | 2 | 1 | 11 | 3 | 11 | 21 | 64 |
| 26 | Tenvis | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 |
| 28 | Totolink | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 2 | 4 | 5 | 5 | 16 | 13 |
| 29 | Trendnet | 1 | 0 | 4 | 2 | 3 | 2 | 7 | 0 | 5 | 8 | 6 | 10 | 13 | 33 |
| 30 | Ubiquiti | 540 | 1,075 | 346 | 293 | 333 | 164 | 198 | 98 | 97 | 78 | 61 | 70 | 97 | 0 |
| 31 | Western-Digital | 2 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 32 | Xiaomi | 13 | 3 | 6 | 6 | 0 | 3 | 23 | 12 | 1 | 62 | 106 | 66 | 7 | 1 |
| 33 | Zyxel | 23 | 17 | 60 | 51 | 41 | 62 | 35 | 82 | 69 | 230 | 120 | 104 | 31 | 46 |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | **Total** | 2,629 | 1,808 | 898 | 1,035 | 1,370 | 707 | 793 | 513 | 501 | 2,325 | 1,144 | 1,044 | 990 | 1125 |

Table A.18: OpenSSL outdated versions for each vendor's firmware image.

| # | Vendor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9-12 | 13-16 | 17-20 | 21-25 | 26-max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 0 | 0 | 0 | 6 | 1 | 1 | 3 | 0 | 0 | 19 | 377 | 21 | 0 | 0 |
| 5 | Alfa | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 3 | 0 | 0 | 0 | 0 |
| 9 | D-Link | 18 | 4 | 8 | 20 | 25 | 8 | 6 | 6 | 0 | 19 | 94 | 8 | 0 | 0 |
| 10 | Dell | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 37 | 4 | 0 | 0 | 0 |
| 12 | EdiMax | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 18 | NETGEAR | 83 | 154 | 379 | 1238 | 973 | 245 | 350 | 62 | 175 | 1343 | 283 | 70 | 0 | 0 |
| 20 | Planet | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | QNAP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 6 | 0 |
| 23 | Synology | 19 | 4 | 4 | 1 | 0 | 6 | 35 | 154 | 5 | 53 | 0 | 0 | 0 | 0 |
| 24 | TP-Link | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 73 | 42 | 0 | 0 |
| 28 | Totolink | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 29 | Trendnet | 0 | 0 | 0 | 1 | 0 | 2 | 2 | 0 | 0 | 2 | 6 | 0 | 0 | 0 |
| 30 | Ubiquiti | 0 | 0 | 2 | 0 | 50 | 46 | 63 | 8 | 7 | 24 | 146 | 1 | 0 | 0 |
| 31 | Western-Digital | 0 | 1 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 32 | Xiaomi | 0 | 0 | 0 | 0 | 16 | 7 | 13 | 14 | 0 | 2 | 28 | 1 | 0 | 0 |
| 33 | Zyxel | 0 | 4 | 9 | 10 | 5 | 15 | 20 | 7 | 3 | 21 | 32 | 0 | 0 | 0 |
| - | **Total** | 121 | 167 | 404 | 1,276 | 1,071 | 334 | 494 | 256 | 190 | 1,533 | 1,044 | 160 | 6 | 0 |

Table A.19: GnuPG outdated versions for each vendor's firmware image.

| # | Vendor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9-12 | 13-16 | 17-20 | 21-25 | 26-max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 0 | 0 | 0 | 0 | 2 | 1 | 1 | 2 | 1 | 0 | 1 | 0 | 0 | 0 |
| 9 | D-Link | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 12 | EdiMax | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
| 18 | NETGEAR | 1048 | 0 | 100 | 1120 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 732 |
| 21 | QNAP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 23 |
| 23 | Synology | 0 | 2 | 17 | 29 | 15 | 19 | 2 | 3 | 16 | 16 | 38 | 1 | 3 | 1 |
| 24 | TP-Link | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 35 | 44 | 13 | 27 | 45 |
| 29 | Trendnet | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 7 | 0 | 0 | 3 |
| 30 | Ubiquiti | 0 | 0 | 0 | 28 | 0 | 7 | 0 | 23 | 56 | 66 | 70 | 58 | 19 | 11 |
| 31 | Western-Digital | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 1 | 0 |
| 33 | Zyxel | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 3 | 5 | 10 | 12 | 5 | 35 |
| - | **Total** | 1,048 | 2 | 117 | 1,178 | 18 | 31 | 8 | 28 | 76 | 122 | 172 | 86 | 55 | 854 |

Table A.20: GnuTLS outdated versions for each vendor's firmware image.

| # | Vendor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9-12 | 13-16 | 17-20 | 21-25 | 26-max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 23 | Synology | 38 | 21 | 4 | 0 | 7 | 50 | 19 | 15 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | TP-Link | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 6 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | Ubiquiti | 0 | 28 | 9 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 31 | Western-Digital | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - | **Total** | 47 | 50 | 13 | 0 | 8 | 50 | 20 | 17 | 0 | 0 | 0 | 0 | 0 | 0 |

Table A.21: Libsodium outdated versions for each vendor's firmware image.

| # | Vendor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9-12 | 13-16 | 17-20 | 21-25 | 26-max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | Alfa | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | Belkin | 0 | 0 | 0 | 0 | 2 | 3 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 9 | D-Link | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 2 | 2 | 2 | 1 | 3 | 1 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 18 | NETGEAR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 6 | 7 | 9 |
| 24 | TP-Link | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 1 | 7 | 0 | 1 | 0 | 0 |
| 30 | Ubiquiti | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 11 | 0 | 37 | 51 | 115 | 0 |
| - | **Total** | 0 | 1 | 2 | 1 | 3 | 14 | 1 | 2 | 14 | 15 | 39 | 60 | 125 | 10 |

Table A.22: WolfSSL outdated versions for each vendor's firmware image.

| # | Vendor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9-12 | 13-16 | 17-20 | 21-25 | 26-max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 4 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 0 | 5 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | NETGEAR | 0 | 1 | 1 | 8 | 6 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | Ubiquiti | 2 | 38 | 60 | 11 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 31 | Western-Digital | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 33 | Zyxel | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - | **Total** | 6 | 48 | 72 | 20 | 9 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table A.23: Nettle outdated versions for each vendor's firmware image.

| # | Vendor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9-12 | 13-16 | 17-20 | 21-25 | 26-max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | D-Link | 86 | 53 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | QNAP | 0 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | Synology | 302 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | Trendnet | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | Ubiquiti | 152 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 32 | Xiaomi | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 33 | Zyxel | 0 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - | **Total** | 556 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table A.24: Mcrypt outdated versions for each vendor's firmware image.

| # | Vendor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9-12 | 13-16 | 17-20 | 21-25 | 26-max |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 2 | 1 | 1 | 0 | 0 | 0 | 0 |
| 12 | EdiMax | 1 | 0 | 0 | 3 | 0 | 5 | 1 | 5 | 1 | 11 | 15 | 15 | 13 | 9 |
| 16 | LinkSys | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | NETGEAR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3 |
| 20 | Planet | 0 | 0 | 2 | 1 | 1 | 0 | 1 | 1 | 0 | 2 | 42 | 0 | 0 | 0 |
| 23 | Synology | 0 | 0 | 2 | 7 | 12 | 16 | 0 | 10 | 9 | 20 | 26 | 45 | 0 | 0 |
| 24 | TP-Link | 1 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 2 | 18 | 27 | 27 | 54 | 48 |
| 28 | Totolink | 0 | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | Trendnet | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 4 |
| 30 | Ubiquiti | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 |
| 32 | Xiaomi | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 33 | Zyxel | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 4 | 0 |
| - | **Total** | 2 | 2 | 4 | 14 | 19 | 24 | 10 | 21 | 13 | 53 | 110 | 89 | 79 | 64 |

Table A.25: MbedTLS outdated versions for each vendor's firmware image.

Table A.26: Overall discovered cryptographic libraries across vendors' firmware images.

| # | Vendors | # unpacked firmwares | GnuTLS | | | # GnuPG (libgcrypt) | | | OpenSSL (libcrypt) | | | WolfSSL | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | # firmwares | % | ✓ | # firmwares | % | ✓ | # firmwares | % | ✓ | # firmwares | % | ✓ |
| 1 | ASUS | 1,309 | 8 | 0.61 | 8 | 428 | 32.70 | 428 | 1,111 | 84.87 | 1,061 | 0 | 0 | 0 |
| 2 | AVM | 102 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 16.67 | 16 | 0 | 0 | 0 |
| 3 | Actiontec | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 100.00 | 5 | 0 | 0 | 0 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 71 | 0 | 0 | 0 | 8 | 11.27 | 8 | 62 | 87.32 | 62 | 1 | 1.41 | 1 |
| 6 | Arris | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 100.00 | 7 | 0 | 0 | 0 |
| 7 | Belkin | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 11.11 | 3 | 10 | 22.22 | 7 |
| 8 | Buffalo | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 100.00 | 4 | 0 | 0 | 0 |
| 9 | D-Link | 2,116 | 4 | 0.19 | 4 | 216 | 10.21 | 216 | 1,694 | 80.06 | 1419 | 19 | 0.90 | 14 |
| 10 | Dell | 122 | 0 | 0 | 0 | 41 | 33.61 | 41 | 41 | 33.61 | 41 | 0 | 0 | 0 |
| 11 | DrayTek | 178 | 0 | 0 | 0 | 0 | 0 | 0 | 175 | 98.31 | 174 | 0 | 0 | 0 |
| 12 | EdiMax | 297 | 4 | 1.35 | 4 | 5 | 1.68 | 5 | 92 | 30.98 | 41 | 0 | 0 | 0 |
| 13 | FOSCAM | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 40.00 | 0 | 0 | 0 | 0 |
| 14 | HP | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | Inmarsat | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 81.82 | 9 | 0 | 0 | 0 |
| 16 | LinkSys | 195 | 0 | 0 | 0 | 0 | 0 | 0 | 170 | 87.18 | 166 | 0 | 0 | 0 |
| 17 | MicroTik | 814 | 0 | 0 | 0 | 0 | 0 | 0 | 814 | 100.00 | 814 | 0 | 0 | 0 |
| 18 | NETGEAR | 8,061 | 3,002 | 37.24 | 3,002 | 5,355 | 66.43 | 5,355 | 6,452 | 80.04 | 6267 | 445 | 5.52 | 27 |
| 19 | Netis | 114 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 6.14 | 5 | 0 | 0 | 0 |
| 20 | Planet | 418 | 0 | 0 | 0 | 3 | 0.72 | 3 | 263 | 62.92 | 241 | 14 | 3.35 | 0 |
| 21 | QNAP | 109 | 23 | 21.10 | 23 | 23 | 21.10 | 23 | 33 | 30.28 | 33 | 0 | 0 | 0 |
| 22 | Rotek | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 100.00 | 1 | 0 | 0 | 0 |
| 23 | Synology | 319 | 162 | 50.78 | 162 | 281 | 88.09 | 281 | 312 | 97.81 | 312 | 0 | 0 | 0 |
| 24 | TP-Link | 2,069 | 164 | 7.93 | 164 | 122 | 5.90 | 122 | 1,196 | 57.81 | 1,191 | 861 | 41.61 | 13 |
| 25 | Tenda | 367 | 0 | 0 | 0 | 0 | 0 | 0 | 132 | 35.97 | 130 | 0 | 0 | 0 |
| 26 | Tenvis | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 57.14 | 4 | 0 | 0 | 0 |
| 27 | Thuraya | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 144 | 0 | 0 | 0 | 3 | 2.08 | 3 | 50 | 34.72 | 48 | 0 | 0 | 0 |
| 29 | Trendnet | 267 | 13 | 4.87 | 13 | 13 | 4.87 | 13 | 128 | 47.94 | 94 | 5 | 1.87 | 0 |
| 30 | Ubiquiti | 3,737 | 338 | 9.04 | 338 | 347 | 9.29 | 347 | 3,450 | 92.32 | 3,450 | 225 | 6.02 | 225 |
| 31 | Western-Digital | 5 | 4 | 80.00 | 4 | 4 | 80.00 | 4 | 5 | 100.00 | 4 | 0 | 0 | 0 |
| 32 | Xiaomi | 313 | 0 | 0 | 0 | 81 | 25.88 | 81 | 310 | 99.04 | 309 | 0 | 0 | 0 |
| 33 | Zyxel | 1,317 | 73 | 5.54 | 73 | 126 | 9.57 | 126 | 989 | 75.09 | 971 | 33 | 2.51 | 0 |
| - | **Total** | 22,548 | 3,795 | 16.83 | 3,795 | 7,056 | 31.29 | 7,056 | 17,540 | 77.79 | 16,882 | 1,613 | 7.15 | 287 |

Table A.27: Overall discovered cryptographic libraries across vendors' firmware images.

| # | Vendor | # unpacked firmwares | Libsodium | | | mbed TLS | | | MCRYPT | | | Nettle | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | # firmwares | % | ✓ | # firmwares | % | ✓ | # firmwares | % | ✓ | # firmwares | % | ✓ |
| 1 | ASUS | 1,309 | 0 | 0 | 0 | 8 | 0.61 | 8 | 0 | 0 | 0 | 8 | 0.61 | 8 |
| 2 | AVM | 102 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | Actiontec | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 71 | 0 | 0 | 0 | 27 | 38.03 | 0 | 0 | 0 | 0 | 8 | 11.27 | 8 |
| 6 | Arris | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | Belkin | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | Buffalo | 4 | 0 | 0 | 0 | 2 | 50.00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | D-Link | 2,116 | 0 | 0 | 0 | 0 | 0 | 0 | 139 | 6.57 | 139 | 0 | 0 | 0 |
| 10 | Dell | 122 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 41 | 33.61 | 0 |
| 11 | DrayTek | 178 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | EdiMax | 297 | 0 | 0 | 0 | 79 | 26.60 | 79 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | FOSCAM | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | HP | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | Inmarsat | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | LinkSys | 195 | 0 | 0 | 0 | 17 | 8.72 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | MicroTik | 814 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | NETGEAR | 8,061 | 0 | 0 | 0 | 57 | 0.71 | 4 | 0 | 0 | 0 | 2909 | 36.09 | 18 |
| 19 | Netis | 114 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | Planet | 418 | 0 | 0 | 0 | 50 | 11.96 | 50 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | QNAP | 109 | 0 | 0 | 0 | 8 | 7.34 | 0 | 23 | 21.10 | 23 | 0 | 0 | 0 |
| 22 | Rotek | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | Synology | 319 | 154 | 48.28 | 154 | 147 | 46.08 | 147 | 302 | 94.67 | 302 | 162 | 50.78 | 0 |
| 24 | TP-Link | 2,069 | 3 | 0.14 | 3 | 185 | 8.94 | 183 | 0 | 0 | 0 | 190 | 9.18 | 0 |
| 25 | Tenda | 367 | 0 | 0 | 0 | 1 | 0.27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | Tenvis | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | Thuraya | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 144 | 9 | 6.25 | 9 | 10 | 6.94 | 6 | 0 | 0 | 0 | 3 | 2.08 | 2 |
| 29 | Trendnet | 267 | 0 | 0 | 0 | 9 | 3.37 | 8 | 2 | 0.75 | 2 | 10 | 3.75 | 0 |
| 30 | Ubiquiti | 3,737 | 38 | 1.02 | 38 | 22 | 0.59 | 5 | 152 | 4.07 | 152 | 329 | 8.80 | 114 |
| 31 | Western-Digital | 5 | 1 | 20.00 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 80.00 | 4 |
| 32 | Xiaomi | 313 | 0 | 0 | 0 | 231 | 73.80 | 4 | 68 | 21.73 | 14 | 0 | 0 | 0 |
| 33 | Zyxel | 1,317 | 0 | 0 | 0 | 6 | 0.46 | 6 | 14 | 1.06 | 14 | 49 | 3.72 | 3 |
| - | **Total** | 22,548 | 205 | 0.91 | 205 | 859 | 3.81 | 504 | 700 | 3.10 | 646 | 3,713 | 16.47 | 157 |

Table A.28: Overall discovered cryptographic libraries across vendors' firmware images.

| # | Vendor | # unpacked firmwares | Crypto++ | | KerberosV5 | | LIBC | | LibTomCrypt | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | # firmwares | % | # firmwares | % | # firmwares | % | # firmwares | % |
| 1 | ASUS | 1,309 | 0 | 0 | 0 | 0 | 1297 | 99.08 | 0 | 0 |
| 2 | AVM | 102 | 0 | 0 | 0 | 0 | 17 | 16.67 | 0 | 0 |
| 3 | Actiontec | 5 | 0 | 0 | 0 | 0 | 5 | 100.00 | 0 | 0 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 71 | 0 | 0 | 0 | 0 | 35 | 49.30 | 0 | 0 |
| 6 | Arris | 7 | 0 | 0 | 0 | 0 | 7 | 100.00 | 0 | 0 |
| 7 | Belkin | 45 | 0 | 0 | 0 | 0 | 44 | 97.78 | 0 | 0 |
| 8 | Buffalo | 4 | 0 | 0 | 0 | 0 | 4 | 100.00 | 0 | 0 |
| 9 | D-Link | 2,116 | 0 | 0 | 212 | 10.02 | 2088 | 98.68 | 44 | 2.08 |
| 10 | Dell | 122 | 0 | 0 | 41 | 33.61 | 96 | 78.69 | 0 | 0 |
| 11 | DrayTek | 178 | 0 | 0 | 0 | 0 | 177 | 99.44 | 0 | 0 |
| 12 | EdiMax | 297 | 0 | 0 | 0 | 0 | 283 | 95.29 | 0 | 0 |
| 13 | FOSCAM | 5 | 0 | 0 | 0 | 0 | 4 | 80.00 | 0 | 0 |
| 14 | HP | 17 | 0 | 0 | 0 | 0 | 17 | 100.00 | 0 | 0 |
| 15 | Inmarsat | 11 | 0 | 0 | 0 | 0 | 9 | 81.82 | 0 | 0 |
| 16 | LinkSys | 195 | 0 | 0 | 0 | 0 | 193 | 98.97 | 0 | 0 |
| 17 | MicroTik | 814 | 0 | 0 | 0 | 0 | 462 | 56.76 | 0 | 0 |
| 18 | NETGEAR | 8,061 | 4106 | 50.94 | 4857 | 60.25 | 7841 | 97.27 | 4 | 0.05 |
| 19 | Netis | 114 | 0 | 0 | 0 | 0 | 114 | 100.00 | 0 | 0 |
| 20 | Planet | 418 | 1 | 0.24 | 9 | 2.15 | 354 | 84.69 | 0 | 0 |
| 21 | QNAP | 109 | 0 | 0 | 23 | 21.10 | 68 | 62.39 | 0 | 0 |
| 22 | Rotek | 1 | 0 | 0 | 0 | 0 | 1 | 100.00 | 0 | 0 |
| 23 | Synology | 319 | 0 | 0 | 312 | 97.81 | 319 | 100.00 | 0 | 0 |
| 24 | TP-Link | 2,069 | 0 | 0 | 0 | 0 | 2059 | 99.52 | 0 | 0 |
| 25 | Tenda | 367 | 0 | 0 | 0 | 0 | 361 | 98.37 | 0 | 0 |
| 26 | Tenvis | 7 | 0 | 0 | 0 | 0 | 7 | 100.00 | 0 | 0 |
| 27 | Thuraya | 2 | 0 | 0 | 0 | 0 | 2 | 100.00 | 0 | 0 |
| 28 | Totolink | 144 | 0 | 0 | 0 | 0 | 144 | 100.00 | 0 | 0 |
| 29 | Trendnet | 267 | 0 | 0 | 0 | 0 | 257 | 96.25 | 0 | 0 |
| 30 | Ubiquiti | 3737 | 0 | 0 | 347 | 9.29 | 3518 | 94.14 | 0 | 0 |
| 31 | Western-Digital | 5 | 0 | 0 | 4 | 80.00 | 5 | 100.00 | 0 | 0 |
| 32 | Xiaomi | 313 | 82 | 26.20 | 0 | 0 | 312 | 99.68 | 0 | 0 |
| 33 | Zyxel | 1317 | 0 | 0 | 24 | 1.82 | 1166 | 88.53 | 0 | 0 |
| - | **Total** | 22,548 | 4,189 | 18.58 | 5,829 | 25.85 | 21,266 | 94.31 | 48 | 0.21 |

Table A.29: Overall discovered cryptographic libraries that reached their End of Life (EoL) span across vendors' firmware images even before the image was released.

| # | Vendor | GnuTLS | | | GnuPG | | | Libsodium | | | MbedTLS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | #firmwares | # EoL | % | #firmwares | # EoL | % | #firmwares | # EoL | % | #firmwares | # EoL | % |
| 1 | Actiontec | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 2 | Addvaluetech | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 3 | Alfa | 0 | 0 | 0.00% | 8 | 8 | 100.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 4 | Arris | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 5 | ASUS | 8 | 0 | 0.00% | 428 | 420 | 98.13% | 0 | 0 | 0.00% | 8 | 0 | 0.00% |
| 6 | AVM | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 7 | Belkin | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 8 | Buffalo | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 9 | D-Link | 4 | 4 | 100.00% | 216 | 137 | 63.43% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 10 | Dell | 0 | 0 | 0.00% | 41 | 41 | 100.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 11 | DrayTek | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 12 | EdiMax | 4 | 0 | 0.00% | 5 | 5 | 100.00% | 0 | 0 | 0.00% | 79 | 63 | 79.75% |
| 13 | FOSCAM | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 14 | HP | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 15 | Inmarsat | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 16 | LinkSys | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 4 | 0 | 0.00% |
| 17 | MicroTik | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 18 | NETGEAR | 3,002 | 171 | 5.70% | 5,355 | 2,511 | 46.89% | 0 | 0 | 0.00% | 4 | 4 | 100.00% |
| 19 | Netis | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 20 | Planet | 0 | 0 | 0.00% | 3 | 0 | 0.00% | 0 | 0 | 0.00% | 50 | 0 | 0.00% |
| 21 | QNAP | 23 | 23 | 100.00% | 23 | 23 | 100.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 22 | Rotek | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 23 | Synology | 162 | 0 | 0.00% | 281 | 253 | 90.04% | 154 | 0 | 0.00% | 147 | 0 | 0.00% |
| 24 | Tenda | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 25 | Tenvis | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 26 | Thuraya | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 27 | Totolink | 0 | 0 | 0.00% | 3 | 3 | 100.00% | 9 | 0 | 0.00% | 6 | 0 | 0.00% |
| 28 | TP-Link | 164 | 2 | 1.22% | 122 | 122 | 100.00% | 3 | 0 | 0.00% | 183 | 172 | 93.99% |
| 29 | Trendnet | 13 | 10 | 76.92% | 13 | 11 | 84.62% | 0 | 0 | 0.00% | 8 | 6 | 75.00% |
| 30 | Ubiquiti | 338 | 185 | 54.73% | 347 | 270 | 77.81% | 38 | 0 | 0.00% | 5 | 5 | 100.00% |
| 31 | Western-Digital | 4 | 0 | 0.00% | 4 | 3 | 75.00% | 1 | 0 | 0.00% | 0 | 0 | 0.00% |
| 32 | Xiaomi | 0 | 0 | 0.00% | 81 | 45 | 55.56% | 0 | 0 | 0.00% | 4 | 0 | 0.00% |
| 33 | Zyxel | 73 | 8 | 10.96% | 126 | 91 | 72.22% | 0 | 0 | 0.00% | 6 | 6 | 100.00% |
| - | Total | 3,795 | 403 | 10.62% | 7056 | 3,943 | 55.88% | 205 | 0 | 0.00% | 504 | 256 | 50.79% |

Table A.30: Overall discovered cryptographic libraries that reached their End of Life (EoL) span across vendors' firmware images even before the image was released.

| # | Vendor | MCRYPT | | | Nettle | | | OpenSSL | | | WolfSSL | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | #firmwares | # EoL | % | #firmwares | # EoL | % | #firmwares | # EoL | % | #firmwares | # EoL | % |
| 1 | Actiontec | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 5 | 5 | 100.00% | 0 | 0 | 0.00% |
| 2 | Addvaluetech | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 3 | Alfa | 0 | 0 | 0.00% | 8 | 0 | 0.00% | 62 | 41 | 66.13% | 1 | 0 | 0.00% |
| 4 | Arris | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 7 | 1 | 14.29% | 0 | 0 | 0.00% |
| 5 | ASUS | 0 | 0 | 0.00% | 8 | 0 | 0.00% | 1061 | 324 | 30.54% | 0 | 0 | 0.00% |
| 6 | AVM | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 16 | 9 | 56.25% | 0 | 0 | 0.00% |
| 7 | Belkin | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 3 | 1 | 33.33% | 7 | 0 | 0.00% |
| 8 | Buffalo | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 4 | 1 | 25.00% | 0 | 0 | 0.00% |
| 9 | D-Link | 139 | 0 | 0.00% | 0 | 0 | 0.00% | 1419 | 645 | 45.45% | 14 | 4 | 28.57% |
| 10 | Dell | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 41 | 7 | 17.07% | 0 | 0 | 0.00% |
| 11 | DrayTek | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 174 | 56 | 32.18% | 0 | 0 | 0.00% |
| 12 | EdiMax | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 41 | 32 | 78.05% | 0 | 0 | 0.00% |
| 13 | FOSCAM | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 14 | HP | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 15 | Inmarsat | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 9 | 6 | 66.67% | 0 | 0 | 0.00% |
| 16 | LinkSys | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 166 | 86 | 51.81% | 0 | 0 | 0.00% |
| 17 | MicroTik | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 814 | 583 | 71.62% | 0 | 0 | 0.00% |
| 18 | NETGEAR | 0 | 0 | 0.00% | 18 | 0 | 0.00% | 6,267 | 2615 | 41.73% | 27 | 16 | 59.26% |
| 19 | Netis | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 5 | 5 | 100.00% | 0 | 0 | 0.00% |
| 20 | Planet | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 241 | 134 | 55.60% | 0 | 0 | 0.00% |
| 21 | QNAP | 23 | 0 | 0.00% | 0 | 0 | 0.00% | 33 | 22 | 66.67% | 0 | 0 | 0.00% |
| 22 | Rotek | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 1 | 1 | 100.00% | 0 | 0 | 0.00% |
| 23 | Synology | 302 | 0 | 0.00% | 0 | 0 | 0.00% | 312 | 45 | 14.42% | 0 | 0 | 0.00% |
| 24 | Tenda | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 130 | 91 | 70.00% | 0 | 0 | 0.00% |
| 25 | Tenvis | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 4 | 0 | 0.00% | 0 | 0 | 0.00% |
| 26 | Thuraya | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 0 | 0 | 0.00% |
| 27 | Totolink | 0 | 0 | 0.00% | 2 | 0 | 0.00% | 48 | 41 | 85.42% | 0 | 0 | 0.00% |
| 28 | TP-Link | 0 | 0 | 0.00% | 0 | 0 | 0.00% | 1,191 | 1,033 | 86.73% | 13 | 0 | 0.00% |
| 29 | Trendnet | 2 | 0 | 0.00% | 0 | 0 | 0.00% | 94 | 55 | 58.51% | 0 | 0 | 0.00% |
| 30 | Ubiquiti | 152 | 0 | 0.00% | 114 | 0 | 0.00% | 3,450 | 696 | 20.17% | 225 | 166 | 73.78% |
| 31 | Western-Digital | 0 | 0 | 0.00% | 4 | 0 | 0.00% | 4 | 3 | 75.00% | 0 | 0 | 0.00% |
| 32 | Xiaomi | 14 | 0 | 0.00% | 0 | 0 | 0.00% | 309 | 186 | 60.19% | 0 | 0 | 0.00% |
| 33 | Zyxel | 14 | 0 | 0.00% | 3 | 0 | 0.00% | 971 | 410 | 42.22% | 0 | 0 | 0.00% |
| - | Total | 646 | 0 | 0.00% | 157 | 0 | 0.00% | 16,882 | 7,134 | 42.26% | 287 | 186 | 64.81% |

Table A.31: Overall analyzed binaries that use OpenSSL cryptographic library by vendor.

| # | Vendor | # analysed binaries | # analysed executables | # analysed libraries | # OpenSSL binaries | % | # OpenSSL libraries | % | # OpenSSL executables | % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 59,484 | 28,076 | 31,408 | 44,090 | 74.12 | 23,678 | 53.70 | 20,412 | 46.30 |
| 2 | AVM | 612 | 245 | 367 | 488 | 79.74 | 194 | 39.75 | 294 | 60.25 |
| 3 | Actiontec | 198 | 5 | 193 | 21 | 10.61 | 2 | 9.52 | 19 | 90.48 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 518 | 209 | 309 | 275 | 53.09 | 153 | 55.64 | 122 | 44.36 |
| 6 | Arris | 309 | 49 | 260 | 250 | 80.91 | 39 | 15.60 | 211 | 84.40 |
| 7 | Belkin | 117 | 24 | 93 | 14 | 11.97 | 12 | 85.71 | 2 | 14.29 |
| 8 | Buffalo | 26 | 6 | 20 | 11 | 42.31 | 0 | 0.00 | 11 | 100.00 |
| 9 | D-Link | 59,426 | 22,688 | 36,738 | 40,121 | 67.51 | 16,790 | 41.85 | 23,331 | 58.15 |
| 10 | Dell | 3,851 | 2,519 | 1,332 | 1,552 | 40.30 | 1,015 | 65.40 | 537 | 34.60 |
| 11 | DrayTek | 3,816 | 556 | 3,260 | 3,604 | 94.44 | 513 | 14.23 | 3,091 | 85.77 |
| 12 | EdiMax | 1,271 | 87 | 1,184 | 278 | 21.87 | 24 | 8.63 | 254 | 91.37 |
| 13 | FOSCAM | 17 | 3 | 14 | 9 | 52.94 | 0 | 0.00 | 9 | 100.00 |
| 14 | HP | 22 | 0 | 22 | 5 | 22.73 | 0 | 0.00 | 5 | 100.00 |
| 15 | Inmarsat | 298 | 69 | 229 | 233 | 78.19 | 46 | 19.74 | 187 | 80.26 |
| 16 | LinkSys | 3,649 | 807 | 2,842 | 1,849 | 50.67 | 444 | 24.01 | 1,405 | 75.99 |
| 17 | MicroTik | 9,374 | 539 | 8,835 | 9,374 | 100.00 | 539 | 5.75 | 8,835 | 94.25 |
| 18 | NETGEAR | 859,333 | 356,226 | 503,107 | 288,404 | 33.56 | 113,933 | 39.50 | 174,471 | 60.50 |
| 19 | Netis | 268 | 5 | 263 | 10 | 3.73 | 0 | 0.00 | 10 | 100.00 |
| 20 | Planet | 6,575 | 1,464 | 5,111 | 3,539 | 53.83 | 1,022 | 28.88 | 2,517 | 71.12 |
| 21 | QNAP | 9,821 | 3,570 | 6,251 | 7,984 | 81.30 | 2,632 | 32.97 | 5,352 | 67.03 |
| 22 | Rotek | 5 | 0 | 5 | 1 | 20.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 228,040 | 112,921 | 115,119 | 160,501 | 70.38 | 90,774 | 56.56 | 69,727 | 43.44 |
| 24 | TP-Link | 27,163 | 7,297 | 19,866 | 8,913 | 32.81 | 2,588 | 29.04 | 6,325 | 70.96 |
| 25 | Tenda | 2,379 | 160 | 2,219 | 357 | 15.01 | 40 | 11.20 | 317 | 88.80 |
| 26 | Tenvis | 129 | 8 | 121 | 7 | 5.43 | 0 | 0.00 | 7 | 100.00 |
| 27 | Thuraya | 10 | 2 | 8 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 28 | Totolink | 1,029 | 241 | 788 | 300 | 29.15 | 146 | 48.67 | 154 | 51.33 |
| 29 | Trendnet | 2,386 | 513 | 1,873 | 1,187 | 49.75 | 263 | 22.16 | 924 | 77.84 |
| 30 | Ubiquiti | 94,850 | 30,526 | 64,324 | 56,496 | 59.56 | 23,805 | 42.14 | 32,691 | 57.86 |
| 31 | Western-Digital | 1349 | 359 | 990 | 581 | 43.07 | 182 | 31.33 | 399 | 68.67 |
| 32 | Xiaomi | 16,059 | 3,932 | 12,127 | 9,003 | 56.06 | 1,579 | 17.54 | 7,424 | 82.46 |
| 33 | Zyxel | 59,655 | 16,987 | 42,668 | 37,720 | 63.23 | 8,310 | 22.03 | 29,410 | 77.97 |
| - | **Total** | 1,452,039 | 590,093 | 861,946 | 677,177 | 45.40 | 288,723 | 42.64 | 388,454 | 57.36 |

Table A.32: Overall analyzed binaries that use GnuPG (libgcrypt) cryptographic library by vendor.

| # | Vendor | # analysed binaries | # analysed executables | # analysed libraries | # GnuPG binaries | % | # GnuPG libraries | % | # GnuPG executables | % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 59,484 | 28,076 | 31,408 | 5,234 | 8.80 | 2,418 | 46.20 | 2,816 | 53.80 |
| 2 | AVM | 612 | 245 | 367 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 3 | Actiontec | 198 | 5 | 193 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 4 | Addvaluetech | 0 | 0 | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 518 | 209 | 309 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 6 | Arris | 309 | 49 | 260 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 7 | Belkin | 117 | 24 | 93 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 8 | Buffalo | 26 | 6 | 20 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 9 | D-Link | 59,426 | 22,688 | 36,738 | 644 | 1.08 | 307 | 47.67 | 337 | 52.33 |
| 10 | Dell | 3,851 | 2,519 | 1,332 | 38 | 0.99 | 38 | 100.00 | 0 | 0.00 |
| 11 | DrayTek | 3,816 | 556 | 3,260 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 12 | EdiMax | 1,271 | 87 | 1,184 | 17 | 1.34 | 5 | 29.41 | 12 | 70.59 |
| 13 | FOSCAM | 17 | 3 | 14 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 14 | HP | 22 | 0 | 22 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 15 | Inmarsat | 298 | 69 | 229 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 16 | LinkSys | 3,649 | 807 | 2,842 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 17 | MicroTik | 9,374 | 539 | 8,835 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 18 | NETGEAR | 859,333 | 356,226 | 503,107 | 209,272 | 24.35 | 67,700 | 32.35 | 141,572 | 67.65 |
| 19 | Netis | 268 | 5 | 263 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 20 | Planet | 6,575 | 1,464 | 5,111 | 5 | 0.08 | 3 | 60.00 | 2 | 40.00 |
| 21 | QNAP | 9,821 | 3,570 | 6,251 | 391 | 3.98 | 253 | 64.71 | 138 | 35.29 |
| 22 | Rotek | 5 | 0 | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 23 | Synology | 228,040 | 112,921 | 115,119 | 23,729 | 10.41 | 17,311 | 72.95 | 6,418 | 27.05 |
| 24 | TP-Link | 27,163 | 7,297 | 19,866 | 109 | 0.40 | 70 | 64.22 | 39 | 35.78 |
| 25 | Tenda | 2,379 | 160 | 2,219 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 26 | Tenvis | 129 | 8 | 121 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 27 | Thuraya | 10 | 2 | 8 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 28 | Totolink | 1,029 | 241 | 788 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 29 | Trendnet | 2,386 | 513 | 1,873 | 27 | 1.13 | 12 | 44.44 | 15 | 55.56 |
| 30 | Ubiquiti | 9,4850 | 30,526 | 64,324 | 10,711 | 11.29 | 1,654 | 15.44 | 9,057 | 84.56 |
| 31 | Western-Digital | 1,349 | 359 | 990 | 466 | 34.54 | 35 | 7.51 | 431 | 92.49 |
| 32 | Xiaomi | 16,059 | 3,932 | 12,127 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 33 | Zyxel | 59,655 | 16,987 | 42,668 | 233 | 0.39 | 169 | 72.53 | 64 | 27.47 |
| - | **Total** | 1,452,039 | 590,093 | 861,946 | 250,876 | 2.99 | 89,975 | 35.86 | 160,901 | 64.14 |

## A.7. Common Vulnerabilities and Exposures (CVEs)

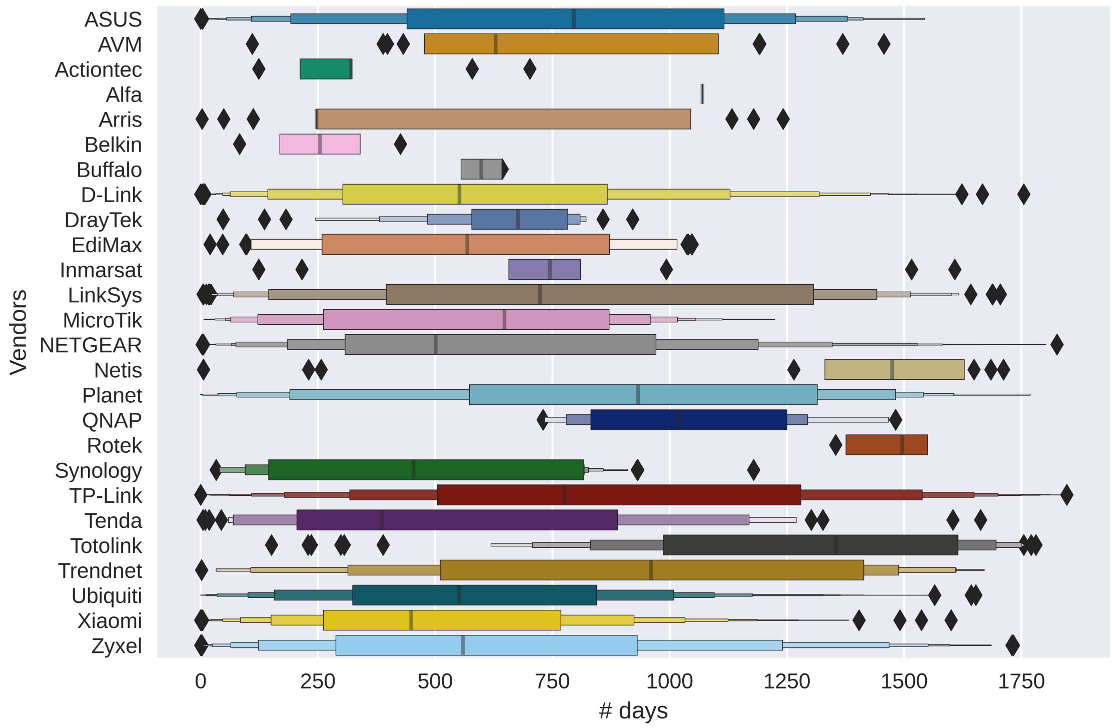



Figure A.11: Critical severity CVEs time-gap in days earlier than firmware release date.

| # | Vendor | #days | # | Vendor | #days |
|---|---|---|---|---|---|
| 1 | ASUS | 766.66 | 18 | NETGEAR | 626.87 |
| 2 | AVM | 780.48 | 19 | Netis | 1,271.96 |
| 3 | Actiontec | 329.11 | 20 | Planet | 899.25 |
| 4 | Addvaluetech | 0 | 21 | QNAP | 1,044.39 |
| 5 | Alfa | 1070 | 22 | Rotek | 1,466.66 |
| 6 | Arris | 493 | 23 | Synology | 470.81 |
| 7 | Belkin | 254.5 | 24 | TP-Link | 870.21 |
| 8 | Buffalo | 598.75 | 25 | Tenda | 565.05 |
| 9 | D-Link | 614.90 | 26 | Tenvis | 0 |
| 10 | Dell | 0 | 27 | Thuraya | 0 |
| 11 | DrayTek | 657.26 | 28 | Totolink | 1,284.23 |
| 12 | EdiMax | 570.19 | 29 | Trendnet | 918.39 |
| 13 | FOSCAM | 0 | 30 | Ubiquiti | 581.64 |
| 14 | HP | 0 | 31 | Western-Digital | 0 |
| 15 | Inmarsat | 784.23 | 32 | Xiaomi | 512.48 |
| 16 | LinkSys | 832.94 | 33 | Zyxel | 638.60 |
| 17 | MicroTik | 583.05 | - | - | - |
| - | Median | 590.47 | - | Mean | 590.47 |

Table A.33: CVE critical severity mean values for each vendor's firmware image.

| # | Vendor | #days | # | Vendor | #days |
|---|---|---|---|---|---|
| 1 | ASUS | 681.68 | 18 | NETGEAR | 765.19 |
| 2 | AVM | 763.27 | 19 | Netis | 1,936.70 |
| 3 | Actiontec | 1334 | 20 | Planet | 1,033.15 |
| 4 | Addvaluetech | 0 | 21 | QNAP | 1,643.70 |
| 5 | Alfa | 1,603.41 | 22 | Rotek | 1,352.25 |
| 6 | Arris | 529.60 | 23 | Synology | 453.17 |
| 7 | Belkin | 310.72 | 24 | TP-Link | 1,057.86 |
| 8 | Buffalo | 1183.90 | 25 | Tenda | 1,323.22 |
| 9 | D-Link | 1057.41 | 26 | Tenvis | 1,187 |
| 10 | Dell | 570.17 | 27 | Thuraya | 0 |
| 11 | DrayTek | 768.67 | 28 | Totolink | 1,517.16 |
| 12 | EdiMax | 829.61 | 29 | Trendnet | 1,223.79 |
| 13 | FOSCAM | 0 | 30 | Ubiquiti | 694.93 |
| 14 | HP | 0 | 31 | Western-Digital | 711.15 |
| 15 | Inmarsat | 1424 | 32 | Xiaomi | 520.64 |
| 16 | LinkSys | 1,038.98 | 33 | Zyxel | 778.12 |
| 17 | MicroTik | 567.68 | - | - | - |
| - | Mean | 874.58 | - | Median | 874.58 |

Table A.34: CVE high severity mean values for each vendor's firmware image.

| Critical | | High | | Medium | | Low | |
|---|---|---|---|---|---|---|---|
| CVE | # Firm. | CVE | # Firm. | CVE | # Firm. | CVE | # Firm. |
| GnuPG [6] (libgcrypt) | | | | | | | |
| - | - | CVE-2017-0379 | 2,753 | CVE-2016-6313 | 3,364 | CVE-2015-7511 | 3,870 |
| - | - | CVE-2018-6829 | 2,165 | CVE-2017-9526 | 3,055 | CVE-2013-4242 | 3,831 |
| - | - | - | - | CVE-2018-0495 | 1,935 | CVE-2014-5270 | 2860 |
| - | - | - | - | CVE-2017-7526 | 1,880 | - | - |
| - | - | - | - | CVE-2015-0837 | 382 | - | - |
| - | - | - | - | CVE-2014-3591 | 382 | - | - |
| - | - | - | - | CVE-2019-12904 | 2 | - | - |
| GnuTLS [7] (libgnutls) | | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| CVE-2017-5336 | 420 | CVE-2014-3468 | 1,778 | CVE-2013-1619 | 1,957 | - | - |
| CVE-2017-5334 | 420 | CVE-2015-3308 | 1,250 | CVE-2014-1959 | 1,931 | - | - |
| CVE-2017-5337 | 420 | CVE-2012-1663 | 777 | CVE-2014-0092 | 1,931 | - | - |
| - | - | CVE-2017-7869 | 686 | CVE-2014-3467 | 1,778 | - | - |
| - | - | CVE-2016-7444 | 668 | CVE-2014-3469 | 1,778 | - | - |
| - | - | CVE-2017-7507 | 594 | CVE-2014-3466 | 1,778 | - | - |
| - | - | CVE-2017-5335 | 420 | CVE-2015-0282 | 1,397 | - | - |
| - | - | CVE-2019-3829 | 110 | CVE-2013-2116 | 1,162 | - | - |
| - | - | CVE-2015-0294 | 43 | CVE-2012-0390 | 779 | - | - |
| - | - | CVE-2009-2730 | 26 | CVE-2012-1573 | 777 | - | - |
| - | - | CVE-2009-1416 | 24 | CVE-2012-1569 | 777 | - | - |
| - | - | CVE-2020-24659 | 12 | CVE-2018-10846 | 403 | - | - |
| - | - | CVE-2020-13777 | 6 | CVE-2018-10844 | 403 | - | - |
| - | - | CVE-2020-11501 | 6 | CVE-2018-10845 | 403 | - | - |
| - | - | - | - | CVE-2018-16868 | 351 | - | - |
| - | - | - | - | CVE-2014-8155 | 239 | - | - |
| - | - | - | - | CVE-2015-6251 | 165 | - | - |
| - | - | - | - | CVE-2015-8313 | 30 | - | - |
| - | - | - | - | CVE-2009-1415 | 26 | - | - |
| - | - | - | - | CVE-2009-1417 | 26 | - | - |
| - | - | - | - | CVE-2009-5138 | 21 | - | - |
| - | - | - | - | CVE-2008-4989 | 2 | - | - |
| - | - | - | - | CVE-2014-8564 | 1 | - | - |
| KerberosV5 [93] (libk5crypto) | | | | | | | |
| - | - | CVE-2003-0028 | 2 | - | - | - | - |
| LibTomCrypt [10] (libtomcrypt) | | | | | | | |
| - | - | - | - | CVE-2018-12437 | 383 | - | - |
| mbedTLS/PolarSSL [11] (libmbedcrypto, libmbedtls, libpolarssl, libmbedx509) | | | | | | | |
| CVE-2017-18187 | 342 | CVE-2018-1000520 | 289 | CVE-2018-0498 | 276 | - | - |
| CVE-2018-0487 | 263 | CVE-2015-1182 | 262 | CVE-2018-0497 | 276 | - | - |
| CVE-2018-0488 | 263 | CVE-2014-9744 | 244 | CVE-2015-5291 | 251 | - | - |
| - | - | CVE-2017-2784 | 214 | CVE-2015-8036 | 251 | - | - |
| - | - | CVE-2017-14032 | 177 | CVE-2019-16910 | 152 | - | - |
| - | - | CVE-2018-9988 | 175 | CVE-2019-18222 | 135 | - | - |
| - | - | CVE-2018-9989 | 175 | CVE-2020-10941 | 129 | - | - |
| - | - | CVE-2014-8628 | 126 | CVE-2014-8627 | 88 | - | - |
| - | - | - | - | CVE-2020-16150 | 57 | - | - |
| - | - | - | - | CVE-2014-4911 | 41 | - | - |
| - | - | - | - | CVE-2013-1621 | 40 | - | - |
| - | - | - | - | CVE-2013-4623 | 40 | - | - |
| - | - | - | - | CVE-2013-5915 | 40 | - | - |
| - | - | - | - | CVE-2013-5914 | 40 | - | - |
| - | - | - | - | CVE-2020-10932 | 11 | - | - |
| Nettle [13] (libnettle) | | | | | | | |
| CVE-2015-8805 | 16 | CVE-2016-6489 | 79 | CVE-2018-16869 | 77 | - | - |
| CVE-2015-8804 | 16 | - | - | - | - | - | - |
| CVE-2015-8803 | 16 | - | - | - | - | - | - |
| OpenSSL [14] (libcrypto, libssl) | | | | | | | |
| CVE-2016-2177 | 5,121 | CVE-2016-2180 | 5,066 | CVE-2016-2178 | 5,121 | CVE-2015-4000 | 2,956 |
| CVE-2016-6303 | 4,906 | CVE-2016-2183 | 4,978 | CVE-2016-6306 | 4,875 | CVE-2011-1945 | 2,435 |
| CVE-2016-2182 | 4,906 | CVE-2016-2181 | 4,906 | CVE-2017-3735 | 3,423 | CVE-2014-3566 | 2,353 |
| CVE-2016-2108 | 2,264 | CVE-2016-6302 | 4,906 | CVE-2013-6449 | 2,826 | CVE-2019-1552 | 2,146 |
| CVE-2016-0705 | 1,805 | CVE-2016-2179 | 4,906 | CVE-2016-2107 | 2,747 | CVE-2013-0169 | 2,022 |
| CVE-2016-0799 | 1,805 | CVE-2016-6304 | 4,875 | CVE-2010-5298 | 2,643 | CVE-2019-1563 | 1,904 |
| CVE-2016-2842 | 1,805 | CVE-2016-2106 | 2,747 | CVE-2012-0884 | 2,610 | CVE-2014-0076 | 1,702 |
| CVE-2020-7043 | 36 | CVE-2016-2109 | 2747 | CVE-2012-1165 | 2609 | CVE-2007-3108 | 1124 |
| - | - | CVE-2016-2176 | 2,747 | CVE-2012-2333 | 2,589 | CVE-2020-1968 | 439 |
| - | - | CVE-2012-2110 | 2,562 | CVE-2014-0195 | 2,573 | CVE-2016-0701 | 423 |
| - | - | CVE-2010-4252 | 2,399 | CVE-2014-0221 | 2,573 | CVE-2004-0975 | 154 |
| - | - | CVE-2015-0292 | 2,205 | CVE-2014-3470 | 2,573 | CVE-2009-0591 | 151 |
| - | - | CVE-2018-0732 | 2,196 | CVE-2017-3738 | 2,561 | CVE-2015-1787 | 9 |
| - | - | CVE-2015-1789 | 2,183 | CVE-2017-3737 | 2,523 | - | - |
| - | - | CVE-2014-8176 | 2,065 | CVE-2018-0737 | 2,408 | - | - |
| - | - | CVE-2015-3194 | 1,841 | CVE-2010-4180 | 2,390 | - | - |
| - | - | CVE-2016-0798 | 1,805 | CVE-2011-4619 | 2,379 | - | - |
| - | - | CVE-2016-0797 | 1,805 | CVE-2011-4108 | 2,379 | - | - |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - | - | CVE-2016-2105 | 1,762 | CVE-2011-4576 | 2,379 | - | - |
| - | - | CVE-2011-4109 | 1,592 | CVE-2012-0027 | 2,379 | - | - |
| - | - | CVE-2017-3731 | 1,541 | CVE-2011-4577 | 2,379 | - | - |
| - | - | CVE-2010-0742 | 1,540 | CVE-2015-3195 | 2,378 | - | - |
| - | - | CVE-2009-3245 | 1,443 | CVE-2013-0166 | 2,358 | - | - |
| - | - | CVE-2014-3567 | 1,417 | CVE-2018-0739 | 2,328 | - | - |
| - | - | CVE-2014-3512 | 1,412 | CVE-2014-3508 | 2,305 | - | - |
| - | - | CVE-2014-0224 | 1,380 | CVE-2014-3507 | 2,305 | - | - |
| - | - | CVE-2010-3864 | 1,334 | CVE-2014-3506 | 2,305 | - | - |
| - | - | CVE-2014-0160 | 954 | CVE-2014-3510 | 2,305 | - | - |
| - | - | CVE-2016-8610 | 923 | CVE-2014-3505 | 2,305 | - | - |
| - | - | CVE-2014-3513 | 850 | CVE-2015-1792 | 2,183 | - | - |
| - | - | CVE-2006-2940 | 807 | CVE-2015-1791 | 2,183 | - | - |
| - | - | CVE-2006-3738 | 789 | CVE-2015-1790 | 2,183 | - | - |
| - | - | CVE-2006-2937 | 789 | CVE-2015-1788 | 2,183 | - | - |
| - | - | CVE-2007-4995 | 443 | CVE-2015-0288 | 2,059 | - | - |
| - | - | CVE-2000-1254 | 240 | CVE-2015-0209 | 2,059 | - | - |
| - | - | CVE-2019-1543 | 75 | CVE-2015-0289 | 2,059 | - | - |
| - | - | CVE-2003-0131 | 32 | CVE-2015-0286 | 2,059 | - | - |
| - | - | CVE-2020-1967 | 28 | CVE-2015-0293 | 2,059 | - | - |
| - | - | CVE-2002-0656 | 18 | CVE-2015-0287 | 2,059 | - | - |
| - | - | CVE-2002-0655 | 18 | CVE-2006-7250 | 2,053 | - | - |
| - | - | CVE-2017-3730 | 13 | CVE-2019-1547 | 1,904 | - | - |
| - | - | CVE-2016-7053 | 13 | CVE-2014-3570 | 1,883 | - | - |
| - | - | CVE-2017-3733 | 13 | CVE-2015-0204 | 1,883 | - | - |
| - | - | CVE-2016-7054 | 13 | CVE-2014-3572 | 1,883 | - | - |
| - | - | CVE-2019-0190 | 2 | CVE-2014-8275 | 1,883 | - | - |
| - | - | - | - | CVE-2014-3571 | 1883 | - | - |
| - | - | - | - | CVE-2011-3210 | 1,870 | - | - |
| - | - | - | - | CVE-2011-1473 | 1,861 | - | - |
| - | - | - | - | CVE-2016-0800 | 1,806 | - | - |
| - | - | - | - | CVE-2016-0702 | 1,805 | - | - |
| - | - | - | - | CVE-2019-1559 | 1,776 | - | - |
| - | - | - | - | CVE-2015-3197 | 1,768 | - | - |
| - | - | - | - | CVE-2018-0734 | 1,767 | - | - |
| - | - | - | - | CVE-2018-5407 | 1,746 | - | - |
| - | - | - | - | CVE-2016-0703 | 1,715 | - | - |
| - | - | - | - | CVE-2016-0704 | 1,715 | - | - |
| - | - | - | - | CVE-2017-3736 | 1,705 | - | - |
| - | - | - | - | CVE-2015-3196 | 1,648 | - | - |
| - | - | - | - | CVE-2015-0206 | 1,611 | - | - |
| - | - | - | - | CVE-2015-0205 | 1,611 | - | - |
| - | - | - | - | CVE-2019-1551 | 1,605 | - | - |
| - | - | - | - | CVE-2016-7055 | 1,563 | - | - |
| - | - | - | - | CVE-2013-6450 | 1,524 | - | - |
| - | - | - | - | CVE-2009-1390 | 1,475 | - | - |
| - | - | - | - | CVE-2009-4355 | 1,469 | - | - |
| - | - | - | - | CVE-2010-0433 | 1,461 | - | - |
| - | - | - | - | CVE-2009-1387 | 1,457 | - | - |
| - | - | - | - | CVE-2009-1377 | 1,455 | - | - |
| - | - | - | - | CVE-2009-1378 | 1,455 | - | - |
| - | - | - | - | CVE-2009-3555 | 1,451 | - | - |
| - | - | - | - | CVE-2009-3766 | 1,446 | - | - |
| - | - | - | - | CVE-2009-3765 | 1,446 | - | - |
| - | - | - | - | CVE-2009-3767 | 1,446 | - | - |
| - | - | - | - | CVE-2009-0590 | 1,432 | - | - |
| - | - | - | - | CVE-2009-0789 | 1,432 | - | - |
| - | - | - | - | CVE-2014-3568 | 1,417 | - | - |
| - | - | - | - | CVE-2014-3511 | 1,412 | - | - |
| - | - | - | - | CVE-2014-3509 | 1,412 | - | - |
| - | - | - | - | CVE-2016-7056 | 1,390 | - | - |
| - | - | - | - | CVE-2014-0198 | 1,390 | - | - |
| - | - | - | - | CVE-2008-5077 | 1,296 | - | - |
| - | - | - | - | CVE-2009-1386 | 1,273 | - | - |
| - | - | - | - | CVE-2011-0014 | 1,259 | - | - |
| - | - | - | - | CVE-2008-7270 | 1,207 | - | - |
| - | - | - | - | CVE-2007-5135 | 1,099 | - | - |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - | - | - | - | CVE-2017-3732 | 1,069 | - | - |
| - | - | - | - | CVE-2011-4354 | 1,034 | - | - |
| - | - | - | - | CVE-2013-4353 | 1,009 | - | - |
| - | - | - | - | CVE-2014-5139 | 925 | - | - |
| - | - | - | - | CVE-2009-2409 | 821 | - | - |
| - | - | - | - | CVE-2006-4339 | 804 | - | - |
| - | - | - | - | CVE-2006-4343 | 789 | - | - |
| - | - | - | - | CVE-2005-2946 | 682 | - | - |
| - | - | - | - | CVE-2005-2969 | 663 | - | - |
| - | - | - | - | CVE-2014-3569 | 540 | - | - |
| - | - | - | - | CVE-2010-0740 | 422 | - | - |
| - | - | - | - | CVE-2015-1794 | 415 | - | - |
| - | - | - | - | CVE-2015-3193 | 415 | - | - |
| - | - | - | - | CVE-2011-3207 | 262 | - | - |
| - | - | - | - | CVE-2008-1678 | 199 | - | - |
| - | - | - | - | CVE-2008-0891 | 195 | - | - |
| - | - | - | - | CVE-2020-1971 | 183 | - | - |
| - | - | - | - | CVE-2005-1797 | 166 | - | - |
| - | - | - | - | CVE-2019-1549 | 134 | - | - |
| - | - | - | - | CVE-2010-1633 | 102 | - | - |
| - | - | - | - | CVE-2012-2686 | 76 | - | - |
| - | - | - | - | CVE-2010-2939 | 69 | - | - |
| - | - | - | - | CVE-2004-0081 | 46 | - | - |
| - | - | - | - | CVE-2004-0112 | 46 | - | - |
| - | - | - | - | CVE-2004-0079 | 46 | - | - |
| - | - | - | - | CVE-2020-7041 | 40 | - | - |
| - | - | - | - | CVE-2020-7042 | 40 | - | - |
| - | - | - | - | CVE-2010-0928 | 34 | - | - |
| - | - | - | - | CVE-2003-0851 | 32 | - | - |
| - | - | - | - | CVE-2003-0147 | 32 | - | - |
| - | - | - | - | CVE-2015-1793 | 28 | - | - |
| - | - | - | - | CVE-2012-0050 | 26 | - | - |
| - | - | - | - | CVE-2003-0078 | 20 | - | - |
| - | - | - | - | CVE-2002-0659 | 18 | - | - |
| - | - | - | - | CVE-2018-0733 | 12 | - | - |
| - | - | - | - | CVE-2018-0735 | 10 | - | - |
| - | - | - | - | CVE-2015-0285 | 9 | - | - |
| - | - | - | - | CVE-2015-0207 | 9 | - | - |
| - | - | - | - | CVE-2015-0208 | 9 | - | - |
| - | - | - | - | CVE-2015-0291 | 9 | - | - |
| - | - | - | - | CVE-2015-0290 | 9 | - | - |
| - | - | - | - | CVE-2009-1379 | 5 | - | - |
| - | - | - | - | CVE-2002-1568 | 2 | - | - |
| WolfSSL [17] (libwolfssl, libcyassl) | | | | | | | |
| CVE-2017-2800 | 272 | CVE-2017-8854 | 273 | CVE-2016-7438 | 277 | - | - |
| CVE-2019-6439 | 183 | CVE-2017-8855 | 273 | CVE-2016-7440 | 277 | - | - |
| CVE-2019-16748 | 150 | CVE-2019-19962 | 101 | CVE-2016-7439 | 277 | - | - |
| CVE-2020-36177 | 5 | CVE-2020-12457 | 8 | CVE-2017-6076 | 276 | - | - |
| - | - | CVE-2020-15309 | 8 | CVE-2017-13099 | 247 | - | - |
| - | - | CVE-2020-11713 | 3 | CVE-2018-12436 | 217 | - | - |
| - | - | CVE-2021-3336 | 3 | CVE-2018-16870 | 184 | - | - |
| - | - | - | - | CVE-2019-13628 | 150 | - | - |
| - | - | - | - | CVE-2019-14317 | 101 | - | - |
| - | - | - | - | CVE-2019-19960 | 101 | - | - |
| - | - | - | - | CVE-2019-19963 | 101 | - | - |
| - | - | - | - | CVE-2020-11735 | 32 | - | - |
| - | - | - | - | CVE-2020-24613 | 8 | - | - |
| - | - | - | - | CVE-2020-24585 | 8 | - | - |

Table A.35: Discovered Critical and High CVE and amount of firmware images earlier than a firmware's release date

| Critical | | High | | Medium | | Low | |
|---|---|---|---|---|---|---|---|
| CVE | # Firm. | CVE | # Firm. | CVE | # Firm. | CVE | # Firm. |
| GnuPG [6] (libgcrypt) | | | | | | | |
| - | - | CVE-2018-6829 | 4819 | CVE-2017-7526 | 5096 | CVE-2015-7511 | 2809 |
| - | - | CVE-2017-0379 | 4223 | CVE-2018-0495 | 5049 | CVE-2014-5270 | 1782 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - | - | - | - | CVE-2015-0837 | 4269 | CVE-2013-4242 | 806 |
| - | - | - | - | CVE-2014-3591 | 4269 | - | - |
| - | - | - | - | CVE-2017-9526 | 3921 | - | - |
| - | - | - | - | CVE-2016-6313 | 3335 | - | - |
| GnuTLS [7] (libgnutls) | | | | | | | |
| CVE-2017-5336 | 3031 | CVE-2020-24659 | 3723 | CVE-2018-16868 | 3378 | - | - |
| CVE-2017-5334 | 3031 | CVE-2015-0294 | 3287 | CVE-2018-10845 | 3326 | - | - |
| CVE-2017-5337 | 3031 | CVE-2017-7507 | 3126 | CVE-2018-10846 | 3326 | - | - |
| - | - | CVE-2017-7869 | 3034 | CVE-2018-10844 | 3326 | - | - |
| - | - | CVE-2017-5335 | 3031 | CVE-2015-8313 | 3243 | - | - |
| - | - | CVE-2016-7444 | 2799 | CVE-2015-0282 | 1876 | - | - |
| - | - | CVE-2015-3308 | 2080 | CVE-2014-3467 | 1496 | - | - |
| - | - | CVE-2014-3468 | 1496 | CVE-2014-3469 | 1496 | - | - |
| - | - | CVE-2019-3829 | 8 | CVE-2014-3466 | 1496 | - | - |
| - | - | CVE-2020-13777 | 8 | CVE-2014-1959 | 1342 | - | - |
| - | - | CVE-2012-1663 | 2 | CVE-2014-0092 | 1342 | - | - |
| - | - | - | - | CVE-2014-8155 | 525 | - | - |
| - | - | - | - | CVE-2013-1619 | 154 | - | - |
| - | - | - | - | CVE-2009-5138 | 5 | - | - |
| - | - | - | - | CVE-2012-1573 | 2 | - | - |
| - | - | - | - | CVE-2012-1569 | 2 | - | - |
| LibTomCrypt [10] (libtomcrypt) | | | | | | | |
| - | - | - | - | CVE-2018-12437 | 31 | - | - |
| mbedTLS/PolarSSL [11] (libmbedcrypto, libmbedtls, libpolarssl, libmbedx509) | | | | | | | |
| CVE-2017-18187 | 119 | CVE-2018-1000520 | 172 | CVE-2020-16150 | 423 | - | - |
| CVE-2018-0488 | 39 | CVE-2018-9988 | 89 | CVE-2020-10941 | 351 | - | - |
| CVE-2018-0487 | 38 | CVE-2018-9989 | 89 | CVE-2019-18222 | 345 | - | - |
| - | - | CVE-2017-2784 | 52 | CVE-2019-16910 | 326 | - | - |
| - | - | CVE-2017-14032 | 17 | CVE-2018-0498 | 193 | - | - |
| - | - | CVE-2014-9744 | 4 | CVE-2018-0497 | 193 | - | - |
| - | - | CVE-2014-8628 | 3 | CVE-2015-5291 | 12 | - | - |
| - | - | CVE-2015-1182 | 1 | CVE-2015-8036 | 12 | - | - |
| Nettle [13] (libnettle) | | | | | | | |
| - | - | - | - | CVE-2018-16869 | 2 | - | - |
| OpenSSL [14] (libcrypto, libssl) | | | | | | | |
| CVE-2020-7043 | 9,758 | CVE-2021-23840 | 5,654 | CVE-2020-7041 | 9,793 | CVE-2020-1968 | 5,054 |
| CVE-2016-2108 | 4,430 | CVE-2016-2106 | 4,734 | CVE-2020-7042 | 9,793 | CVE-2015-4000 | 3,685 |
| CVE-2016-6303 | 2,646 | CVE-2016-2109 | 4,734 | CVE-2016-7056 | 8,404 | CVE-2019-1563 | 3,517 |
| CVE-2016-2182 | 2,646 | CVE-2016-2176 | 4,734 | CVE-2021-23841 | 5,654 | CVE-2019-1552 | 3,275 |
| CVE-2016-2177 | 2,431 | CVE-2014-8176 | 3,224 | CVE-2020-1971 | 5,489 | CVE-2014-3566 | 2,274 |
| CVE-2016-0705 | 2,129 | CVE-2015-0292 | 3,077 | CVE-2017-3735 | 4,757 | CVE-2014-0076 | 1,317 |
| CVE-2016-0799 | 2,129 | CVE-2016-6304 | 2,647 | CVE-2016-2107 | 4,734 | CVE-2021-23839 | 649 |
| CVE-2016-2842 | 2,129 | CVE-2016-2181 | 2,646 | CVE-2019-1551 | 3,824 | CVE-2013-0169 | 391 |
| - | - | CVE-2016-6302 | 2,646 | CVE-2019-1547 | 3,517 | CVE-2011-1945 | 218 |
| - | - | CVE-2016-2179 | 2,646 | CVE-2019-1559 | 2,669 | CVE-2007-3108 | 15 |
| - | - | CVE-2016-2183 | 2,523 | CVE-2016-6306 | 2,647 | CVE-2016-0701 | 3 |
| - | - | CVE-2016-2180 | 2,486 | CVE-2014-0195 | 2,457 | - | - |
| - | - | CVE-2016-2105 | 2,203 | CVE-2014-0221 | 2,457 | - | - |
| - | - | CVE-2016-0798 | 2,129 | CVE-2014-3470 | 2,457 | - | - |
| - | - | CVE-2016-0797 | 2,129 | CVE-2016-2178 | 2,431 | - | - |
| - | - | CVE-2015-1789 | 2,043 | CVE-2015-3195 | 2,412 | - | - |
| - | - | CVE-2015-3194 | 1,894 | CVE-2010-5298 | 2,314 | - | - |
| - | - | CVE-2018-0732 | 1,731 | CVE-2018-5407 | 2,230 | - | - |
| - | - | CVE-2014-3567 | 1,288 | CVE-2015-3196 | 2,228 | - | - |
| - | - | CVE-2014-3512 | 1,188 | CVE-2018-0734 | 2,211 | - | - |
| - | - | CVE-2014-0224 | 1,162 | CVE-2016-0702 | 2,129 | - | - |
| - | - | CVE-2014-3513 | 1,157 | CVE-2016-0800 | 2,128 | - | - |
| - | - | CVE-2014-0160 | 903 | CVE-2016-0703 | 2,115 | - | - |
| - | - | CVE-2012-2110 | 390 | CVE-2016-0704 | 2,115 | - | - |
| - | - | CVE-2016-8610 | 368 | CVE-2015-3197 | 2,103 | - | - |
| - | - | CVE-2017-3731 | 197 | CVE-2014-3508 | 2,085 | - | - |
| - | - | CVE-2010-4252 | 165 | CVE-2014-3507 | 2,085 | - | - |
| - | - | CVE-2000-1254 | 158 | CVE-2014-3506 | 2,085 | - | - |
| - | - | CVE-2010-0742 | 128 | CVE-2014-3510 | 2,085 | - | - |
| - | - | CVE-2009-3245 | 105 | CVE-2014-3505 | 2,085 | - | - |
| - | - | CVE-2011-4109 | 102 | CVE-2015-1792 | 2,043 | - | - |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - | - | CVE-2019-1543 | 15 | CVE-2015-1791 | 2,043 | - | - |
| - | - | CVE-2010-3864 | 9 | CVE-2015-1790 | 2,043 | - | - |
| - | - | CVE-2006-2940 | 5 | CVE-2015-1788 | 2,043 | - | - |
| - | - | CVE-2006-2937 | 3 | CVE-2013-6449 | 2,007 | - | - |
| - | - | CVE-2006-3738 | 3 | CVE-2015-0288 | 1,771 | - | - |
| - | - | CVE-2016-7053 | 1 | CVE-2015-0209 | 1,771 | - | - |
| - | - | CVE-2017-3733 | 1 | CVE-2015-0289 | 1,771 | - | - |
| - | - | CVE-2017-3730 | 1 | CVE-2015-0286 | 1,771 | - | - |
| - | - | CVE-2016-7054 | 1 | CVE-2015-0293 | 1,771 | - | - |
| - | - | - | - | CVE-2015-0287 | 1,771 | - | - |
| - | - | - | - | CVE-2014-3570 | 1,681 | - | - |
| - | - | - | - | CVE-2015-0204 | 1,681 | - | - |
| - | - | - | - | CVE-2014-3572 | 1,681 | - | - |
| - | - | - | - | CVE-2014-8275 | 1,681 | - | - |
| - | - | - | - | CVE-2014-3571 | 1,681 | - | - |
| - | - | - | - | CVE-2015-0206 | 1,672 | - | - |
| - | - | - | - | CVE-2015-0205 | 1,672 | - | - |
| - | - | - | - | CVE-2018-0737 | 1,480 | - | - |
| - | - | - | - | CVE-2018-0739 | 1,426 | - | - |
| - | - | - | - | CVE-2014-3568 | 1,288 | - | - |
| - | - | - | - | CVE-2014-3511 | 1,188 | - | - |
| - | - | - | - | CVE-2014-3509 | 1,188 | - | - |
| - | - | - | - | CVE-2014-0198 | 1,152 | - | - |
| - | - | - | - | CVE-2014-5139 | 1,070 | - | - |
| - | - | - | - | CVE-2017-3738 | 981 | - | - |
| - | - | - | - | CVE-2017-3737 | 961 | - | - |
| - | - | - | - | CVE-2013-6450 | 894 | - | - |
| - | - | - | - | CVE-2013-4353 | 825 | - | - |
| - | - | - | - | CVE-2017-3736 | 823 | - | - |
| - | - | - | - | CVE-2013-0166 | 708 | - | - |
| - | - | - | - | CVE-2011-1473 | 513 | - | - |
| - | - | - | - | CVE-2012-2333 | 415 | - | - |
| - | - | - | - | CVE-2012-1165 | 354 | - | - |
| - | - | - | - | CVE-2012-0884 | 353 | - | - |
| - | - | - | - | CVE-2006-7250 | 341 | - | - |
| - | - | - | - | CVE-2011-4619 | 315 | - | - |
| - | - | - | - | CVE-2011-4108 | 315 | - | - |
| - | - | - | - | CVE-2011-4576 | 315 | - | - |
| - | - | - | - | CVE-2012-0027 | 315 | - | - |
| - | - | - | - | CVE-2011-4577 | 315 | - | - |
| - | - | - | - | CVE-2011-4354 | 300 | - | - |
| - | - | - | - | CVE-2014-3569 | 228 | - | - |
| - | - | - | - | CVE-2016-7055 | 205 | - | - |
| - | - | - | - | CVE-2017-3732 | 195 | - | - |
| - | - | - | - | CVE-2008-7270 | 165 | - | - |
| - | - | - | - | CVE-2010-4180 | 165 | - | - |
| - | - | - | - | CVE-2010-0433 | 105 | - | - |
| - | - | - | - | CVE-2009-4355 | 101 | - | - |
| - | - | - | - | CVE-2009-3555 | 93 | - | - |
| - | - | - | - | CVE-2009-3766 | 93 | - | - |
| - | - | - | - | CVE-2009-3765 | 93 | - | - |
| - | - | - | - | CVE-2009-3767 | 93 | - | - |
| - | - | - | - | CVE-2011-3210 | 87 | - | - |
| - | - | - | - | CVE-2009-1390 | 69 | - | - |
| - | - | - | - | CVE-2009-1387 | 65 | - | - |
| - | - | - | - | CVE-2009-1386 | 65 | - | - |
| - | - | - | - | CVE-2009-1377 | 62 | - | - |
| - | - | - | - | CVE-2009-1378 | 62 | - | - |
| - | - | - | - | CVE-2009-0590 | 53 | - | - |
| - | - | - | - | CVE-2009-0789 | 53 | - | - |
| - | - | - | - | CVE-2008-5077 | 42 | - | - |
| - | - | - | - | CVE-2009-2409 | 23 | - | - |
| - | - | - | - | CVE-2018-0735 | 11 | - | - |
| - | - | - | - | CVE-2007-5135 | 7 | - | - |
| - | - | - | - | CVE-2018-0733 | 7 | - | - |
| - | - | - | - | CVE-2010-0740 | 5 | - | - |
| - | - | - | - | CVE-2006-4339 | 4 | - | - |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - | - | - | - | CVE-2006-4343 | 3 | - | - |
| - | - | - | - | CVE-2019-1549 | 2 | - | - |
| - | - | - | - | CVE-2005-2946 | 1 | - | - |
| - | - | - | - | CVE-2011-3207 | 1 | - | - |
| - | - | - | - | CVE-2011-0014 | 1 | - | - |
| - | - | - | - | CVE-2015-1794 | 1 | - | - |
| - | - | - | - | CVE-2015-3193 | 1 | - | - |
| WolfSSL [17] (libwolfssl, libcyassl) | | | | | | | |
| CVE-2020-36177 | 275 | CVE-2021-3336 | 277 | CVE-2020-24585 | 272 | - | - |
| CVE-2019-16748 | 127 | CVE-2020-12457 | 272 | CVE-2020-24613 | 272 | - | - |
| CVE-2019-6439 | 94 | CVE-2020-15309 | 272 | CVE-2020-11735 | 248 | - | - |
| CVE-2017-2800 | 5 | CVE-2019-19962 | 176 | CVE-2019-14317 | 176 | - | - |
| - | - | CVE-2017-8854 | 4 | CVE-2019-19960 | 176 | - | - |
| - | - | CVE-2017-8855 | 4 | CVE-2019-19963 | 176 | - | - |
| - | - | - | - | CVE-2019-13628 | 127 | - | - |
| - | - | - | - | CVE-2018-16870 | 93 | - | - |
| - | - | - | - | CVE-2018-12436 | 60 | - | - |
| - | - | - | - | CVE-2017-13099 | 30 | - | - |
| - | - | - | - | CVE-2017-6076 | 1 | - | - |

Table A.36: Discovered Critical and High CVE and amount of firmware images earlier than a firmware's release date

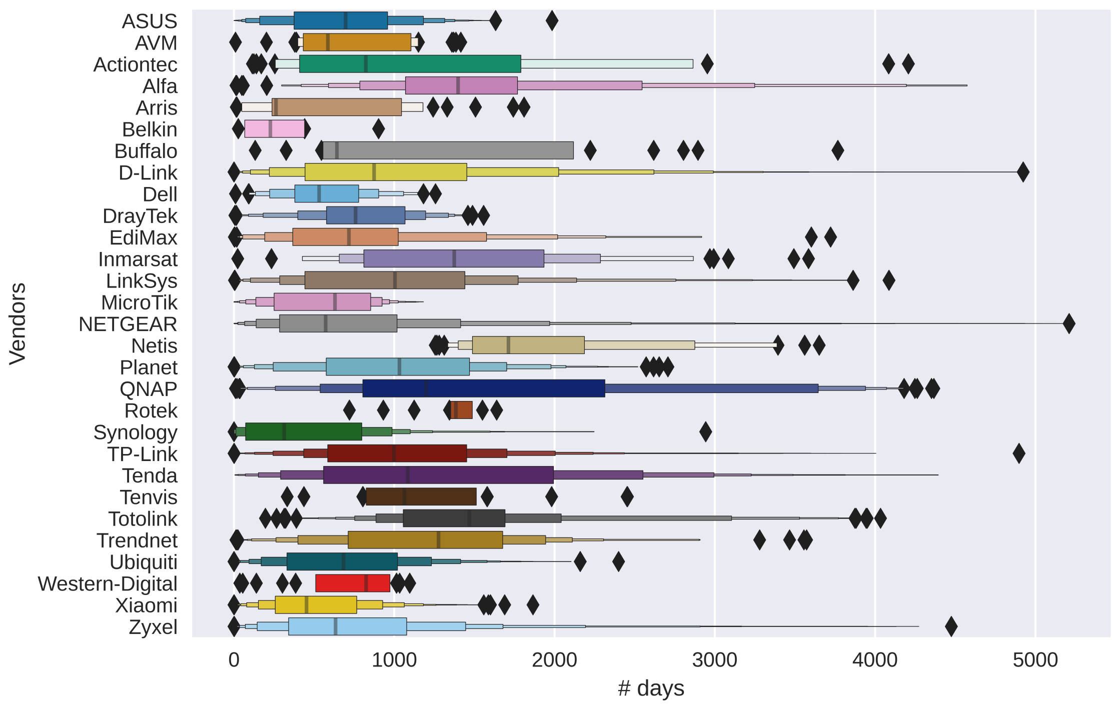



Figure A.12: High severity CVEs time-gap in days earlier than firmware release date.

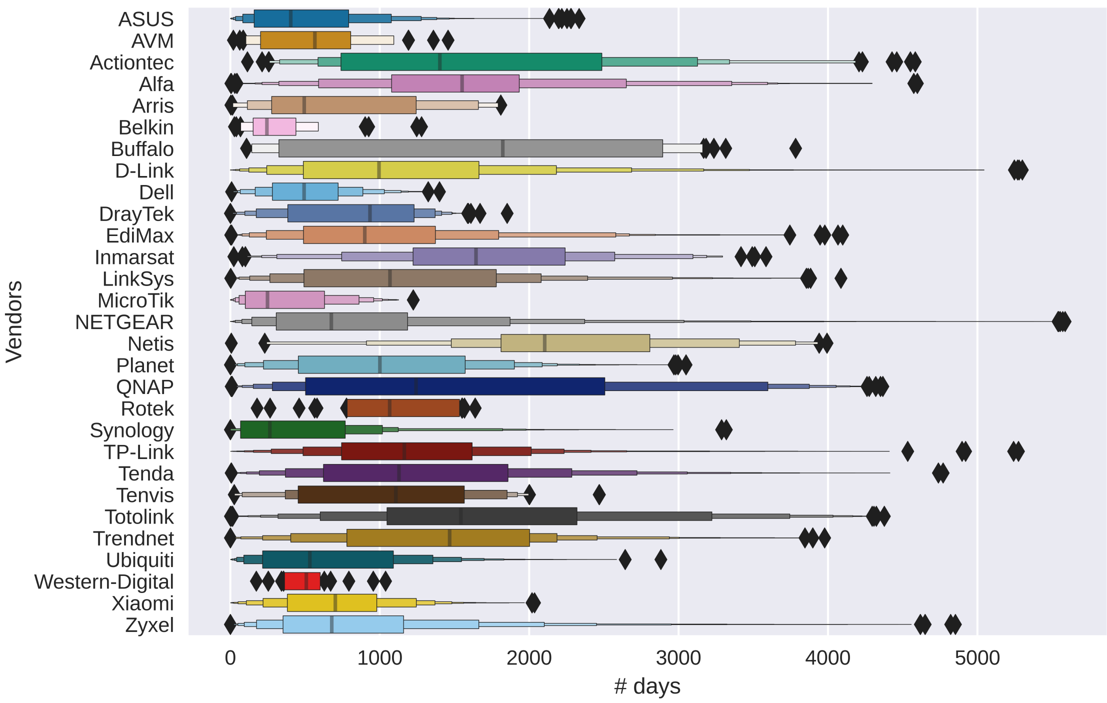



Figure A.13: Medium severity CVEs time-gap in days earlier than firmware release date.

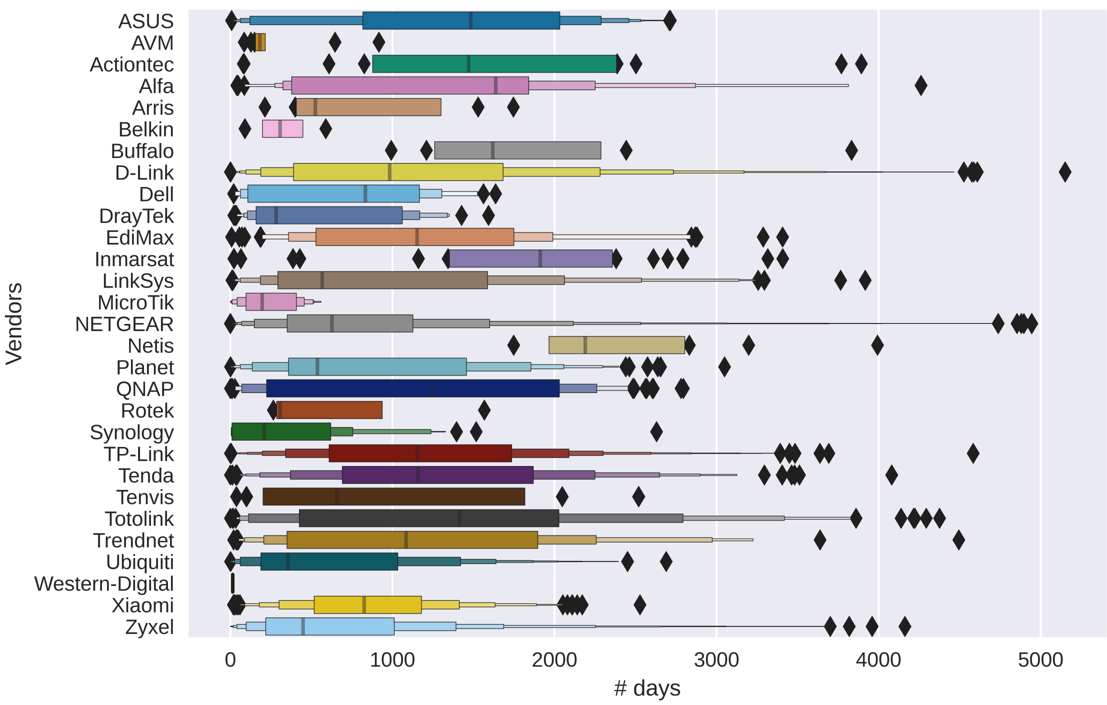



Figure A.14: Low severity CVEs time-gap in days earlier than firmware release date.

Table A.37: Overall distinct CVEs found on firmware images for cryptographic libraries by severity.

| | | Earlier | | | | | | | | Later | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Vendors | Critical | | High | | Medium | | Low | | Critical | | High | | Medium | | Low | |
| | | # CVEs | # Firm. | # CVEs | # Firm. | # CVEs | # Firm. | # CVEs | # Firm. | # CVEs | # Firm. | # CVEs | # Firm. | # CVEs | # Firm. | # CVEs | # Firm. |
| 1 | ASUS | 7 | 195 | 38 | 790 | 114 | 863 | 13 | 742 | 8 | 137 | 33 | 866 | 101 | 897 | 10 | 828 |
| 2 | AVM | 7 | 6 | 16 | 6 | 20 | 10 | 4 | 5 | 1 | 5 | 2 | 5 | 11 | 10 | 3 | 5 |
| 3 | Actiontec | 7 | 3 | 28 | 5 | 81 | 5 | 6 | 5 | 2 | 5 | 6 | 2 | 22 | 5 | 2 | 2 |
| 4 | Addvaluet. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | Alfa | 1 | 25 | 25 | 53 | 90 | 54 | 10 | 45 | 2 | 26 | 4 | 29 | 10 | 54 | 3 | 24 |
| 6 | Arris | 7 | 6 | 18 | 7 | 40 | 7 | 2 | 3 | 8 | 3 | 15 | 6 | 24 | 7 | 3 | 4 |
| 7 | Belkin | 1 | 2 | 16 | 5 | 44 | 5 | 3 | 2 | 11 | 5 | 26 | 5 | 69 | 5 | 6 | 2 |
| 8 | Buffalo | 4 | 2 | 19 | 4 | 50 | 4 | 6 | 1 | 1 | 1 | 2 | 3 | 12 | 4 | 3 | 3 |
| 9 | D-Link | 14 | 464 | 59 | 1,352 | 149 | 1377 | 15 | 1,188 | 12 | 1,102 | 50 | 1,089 | 121 | 1,376 | 12 | 915 |
| 10 | Dell | 0 | 0 | 4 | 41 | 16 | 41 | 4 | 41 | 0 | 0 | 1 | 41 | 8 | 41 | 3 | 40 |
| 11 | DrayTek | 7 | 38 | 29 | 120 | 87 | 171 | 7 | 119 | 8 | 45 | 19 | 135 | 51 | 173 | 6 | 87 |
| 12 | EdiMax | 8 | 35 | 40 | 91 | 112 | 87 | 11 | 63 | 12 | 80 | 45 | 84 | 109 | 96 | 10 | 20 |
| 13 | FOSCAM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | HP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | Inmarsat | 4 | 5 | 20 | 6 | 66 | 7 | 8 | 5 | 1 | 3 | 1 | 4 | 10 | 7 | 3 | 3 |
| 16 | LinkSys | 8 | 121 | 40 | 151 | 113 | 158 | 11 | 111 | 8 | 77 | 24 | 123 | 68 | 158 | 8 | 92 |
| 17 | MicroTik | 6 | 488 | 16 | 551 | 36 | 754 | 5 | 287 | 8 | 569 | 15 | 347 | 28 | 781 | 4 | 219 |
| 18 | NETGEAR | 20 | 2,991 | 63 | 5,188 | 169 | 6,189 | 16 | 5,652 | 14 | 5,520 | 46 | 5,161 | 130 | 6,197 | 14 | 3,332 |
| 19 | Netis | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | Planet | 10 | 104 | 39 | 189 | 113 | 216 | 12 | 188 | 10 | 147 | 35 | 166 | 87 | 216 | 12 | 128 |
| 21 | QNAP | 3 | 21 | 14 | 21 | 41 | 27 | 6 | 25 | 1 | 2 | 3 | 25 | 9 | 27 | 4 | 22 |
| 22 | Rotek | 6 | 1 | 16 | 1 | 22 | 1 | 3 | 1 | 0 | 0 | 1 | 1 | 2 | 1 | 1 | 1 |
| 23 | Synology | 13 | 125 | 42 | 302 | 95 | 291 | 11 | 168 | 14 | 188 | 31 | 305 | 61 | 305 | 6 | 298 |
| 24 | TP-Link | 15 | 818 | 55 | 970 | 144 | 1,023 | 14 | 950 | 18 | 804 | 44 | 517 | 97 | 1,009 | 9 | 220 |
| 25 | Tenda | 7 | 65 | 34 | 124 | 99 | 128 | 10 | 118 | 8 | 110 | 20 | 68 | 53 | 128 | 7 | 46 |
| 26 | Tenvis | 0 | 0 | 6 | 4 | 36 | 4 | 4 | 4 | 2 | 4 | 5 | 4 | 15 | 4 | 3 | 4 |
| 27 | Thuraya | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | Totolink | 7 | 33 | 33 | 35 | 110 | 42 | 12 | 42 | 1 | 13 | 2 | 29 | 11 | 42 | 3 | 11 |
| 29 | Trendnet | 14 | 35 | 55 | 74 | 138 | 76 | 14 | 69 | 11 | 58 | 39 | 56 | 99 | 76 | 12 | 34 |
| 30 | Ubiquiti | 17 | 474 | 50 | 1,377 | 118 | 2,902 | 12 | 1003 | 14 | 497 | 39 | 2,949 | 95 | 2,912 | 11 | 2,402 |
| 31 | WD | 0 | 0 | 8 | 4 | 10 | 4 | 1 | 1 | 0 | 0 | 3 | 4 | 3 | 2 | 1 | 1 |
| 32 | Xiaomi | 10 | 241 | 18 | 282 | 44 | 297 | 4 | 281 | 8 | 297 | 19 | 121 | 44 | 297 | 3 | 36 |
| 33 | Zyxel | 13 | 475 | 55 | 925 | 142 | 960 | 15 | 695 | 11 | 489 | 43 | 829 | 119 | 978 | 11 | 669 |
| - | **Total** | 21 | 6,773 | 78 | 12,678 | 189 | 15,703 | 16 | 11,814 | 18 | 10,187 | 67 | 12,974 | 148 | 15,808 | 14 | 9,448 |

## A.8. Cryptographic Misuses

| # | Vendor | # unpacked firmwares | S1 | % | S2 | % | S3 | % | S4 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 0 | 0.00 | 546 | 41.71 | 0 | 0.00 | 539 | 41.18 | 763 | 58.29 |
| 2 | AVM | 102 | 0 | 0.00 | 5 | 4.90 | 0 | 0.00 | 5 | 4.90 | 97 | 95.10 |
| 3 | Actiontec | 5 | 0 | 0.00 | 1 | 20.00 | 0 | 0.00 | 1 | 20.00 | 4 | 80.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 10 | 14.08 | 0 | 0.00 | 54 | 76.06 | 17 | 23.94 |
| 6 | Arris | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 203 | 9.59 | 356 | 16.82 | 9 | 0.43 | 219 | 10.35 | 1,587 | 75.00 |
| 10 | Dell | 122 | 0 | 0.00 | 41 | 33.61 | 0 | 0.00 | 41 | 33.61 | 81 | 66.39 |
| 11 | DrayTek | 178 | 83 | 46.63 | 5 | 2.81 | 0 | 0.00 | 9 | 5.06 | 91 | 51.12 |
| 12 | EdiMax | 297 | 0 | 0.00 | 4 | 1.35 | 0 | 0.00 | 4 | 1.35 | 293 | 98.65 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 3 | 27.27 | 0 | 0.00 | 3 | 27.27 | 8 | 72.73 |
| 16 | LinkSys | 195 | 2 | 1.03 | 86 | 44.10 | 0 | 0.00 | 99 | 50.77 | 94 | 48.21 |
| 17 | MicroTik | 814 | 0 | 0.00 | 190 | 23.34 | 0 | 0.00 | 0 | 0.00 | 624 | 76.66 |
| 18 | NETGEAR | 8,061 | 183 | 2.27 | 1,714 | 21.26 | 12 | 0.15 | 610 | 7.57 | 6,321 | 78.41 |
| 19 | Netis | 114 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 114 | 100.00 |
| 20 | Planet | 418 | 0 | 0.00 | 60 | 14.35 | 0 | 0.00 | 86 | 20.57 | 332 | 79.43 |
| 21 | QNAP | 109 | 10 | 9.17 | 23 | 21.10 | 0 | 0.00 | 23 | 21.10 | 86 | 78.90 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 0 | 0.00 | 165 | 51.72 | 0 | 0.00 | 196 | 61.44 | 123 | 38.56 |
| 24 | TP-Link | 2,069 | 70 | 3.38 | 162 | 7.83 | 0 | 0.00 | 137 | 6.62 | 1,854 | 89.61 |
| 25 | Tenda | 367 | 0 | 0.00 | 3 | 0.82 | 0 | 0.00 | 22 | 5.99 | 344 | 93.73 |
| 26 | Tenvis | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 4 | 2.78 | 0 | 0.00 | 4 | 2.78 | 140 | 97.22 |
| 29 | Trendnet | 267 | 0 | 0.00 | 34 | 12.73 | 0 | 0.00 | 29 | 10.86 | 228 | 85.39 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 259 | 6.93 | 0 | 0.00 | 387 | 10.36 | 3,350 | 89.64 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 4 | 80.00 | 0 | 0.00 | 4 | 80.00 | 1 | 20.00 |
| 32 | Xiaomi | 313 | 18 | 5.75 | 14 | 4.47 | 0 | 0.00 | 14 | 4.47 | 281 | 89.78 |
| 33 | Zyxel | 1,317 | 0 | 0.00 | 105 | 7.97 | 0 | 0.00 | 344 | 26.12 | 968 | 73.50 |
| - | **Total** | 22,548 | 569 | 2.52 | 3,794 | 16.83 | 21 | 0.09 | 2,830 | 12.55 | 17,889 | 79.34 |

Table A.38: Symmetric Key Cryptography overall cryptographic misuses (found from entry and possible $\phi$).

| # | Vendor | # unpacked firmwares | P1 | % | P2 | % | P3 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 347 | 26.51 | 0 | 0.00 | 1 | 0.08 | 961 | 73.41 |
| 2 | AVM | 102 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 102 | 100.00 |
| 3 | Actiontec | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 3 | 4.23 | 0 | 0.00 | 0 | 0.00 | 68 | 95.77 |
| 6 | Arris | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 25 | 1.18 | 0 | 0.00 | 167 | 7.89 | 1,924 | 90.93 |
| 10 | Dell | 122 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 122 | 100.00 |
| 11 | DrayTek | 178 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 178 | 100.00 |
| 12 | EdiMax | 297 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 297 | 100.00 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 11 | 100.00 |
| 16 | LinkSys | 195 | 1 | 0.51 | 0 | 0.00 | 2 | 1.03 | 192 | 98.46 |
| 17 | MicroTik | 814 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 814 | 100.00 |
| 18 | NETGEAR | 8,061 | 3,197 | 39.66 | 0 | 0.00 | 1,109 | 13.76 | 4,618 | 57.29 |
| 19 | Netis | 114 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 114 | 100.00 |
| 20 | Planet | 418 | 0 | 0.00 | 0 | 0.00 | 24 | 5.74 | 394 | 94.26 |
| 21 | QNAP | 109 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 109 | 100.00 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 23 | Synology | 319 | 69 | 21.63 | 0 | 0.00 | 108 | 33.86 | 144 | 45.14 |
| 24 | TP-Link | 2,069 | 60 | 2.90 | 0 | 0.00 | 63 | 3.04 | 1,958 | 94.64 |
| 25 | Tenda | 367 | 3 | 0.82 | 0 | 0.00 | 0 | 0.00 | 364 | 99.18 |
| 26 | Tenvis | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 1 | 0.69 | 0 | 0.00 | 0 | 0.00 | 143 | 99.31 |
| 29 | Trendnet | 267 | 0 | 0.00 | 0 | 0.00 | 1 | 0.37 | 266 | 99.63 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 0 | 0.00 | 131 | 3.51 | 3,606 | 96.49 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 0 | 0.00 | 4 | 80.00 | 1 | 20.00 |
| 32 | Xiaomi | 313 | 96 | 30.67 | 0 | 0.00 | 14 | 4.47 | 203 | 64.86 |
| 33 | Zyxel | 1,317 | 48 | 3.64 | 0 | 0.00 | 0 | 0.00 | 1,269 | 96.36 |
| - | **Total** | 22,548 | 3,850 | 17.07 | 0 | 0.00 | 1,624 | 7.20 | 17,951 | 79.61 |

Table A.39: Public Key Cryptography overall cryptographic misuses (found from entry and possibly $\phi$).

| # | Vendor | # unpacked firmwares | R1 | % | R2 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 137 | 10.47 | 908 | 69.37 | 353 | 26.97 |
| 2 | AVM | 102 | 0 | 0.00 | 12 | 11.76 | 90 | 88.24 |
| 3 | Actiontec | 5 | 1 | 20.00 | 5 | 100.00 | 0 | 0.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 0 | 0.00 | 71 | 100.00 |
| 6 | Arris | 7 | 0 | 0.00 | 4 | 57.14 | 3 | 42.86 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 2 | 50.00 | 0 | 0.00 | 2 | 50.00 |
| 9 | D-Link | 2,116 | 62 | 2.93 | 950 | 44.90 | 1,140 | 53.88 |
| 10 | Dell | 122 | 0 | 0.00 | 20 | 16.39 | 102 | 83.61 |
| 11 | DrayTek | 178 | 0 | 0.00 | 99 | 55.62 | 79 | 44.38 |
| 12 | EdiMax | 297 | 0 | 0.00 | 22 | 7.41 | 275 | 92.59 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 1 | 20.00 | 4 | 80.00 |
| 14 | HP | 17 | 0 | 0.00 | 5 | 29.41 | 12 | 70.59 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 5 | 45.45 | 6 | 54.55 |
| 16 | LinkSys | 195 | 30 | 15.38 | 117 | 60.00 | 71 | 36.41 |
| 17 | MicroTik | 814 | 0 | 0.00 | 632 | 77.64 | 182 | 22.36 |
| 18 | NETGEAR | 8,061 | 2232 | 27.69 | 4374 | 54.26 | 1,880 | 23.32 |
| 19 | Netis | 114 | 0 | 0.00 | 7 | 6.14 | 107 | 93.86 |
| 20 | Planet | 418 | 4 | 0.96 | 162 | 38.76 | 256 | 61.24 |
| 21 | QNAP | 109 | 0 | 0.00 | 27 | 24.77 | 82 | 75.23 |
| 22 | Rotek | 1 | 0 | 0.00 | 1 | 100.00 | 0 | 0.00 |
| 23 | Synology | 319 | 48 | 15.05 | 200 | 62.70 | 119 | 37.30 |
| 24 | TP-Link | 2,069 | 86 | 4.16 | 1,202 | 58.10 | 845 | 40.84 |
| 25 | Tenda | 367 | 0 | 0.00 | 92 | 25.07 | 275 | 74.93 |
| 26 | Tenvis | 7 | 0 | 0.00 | 2 | 28.57 | 5 | 71.43 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 9 | 6.25 | 135 | 93.75 |
| 29 | Trendnet | 267 | 2 | 0.75 | 68 | 25.47 | 197 | 73.78 |
| 30 | Ubiquiti | 3,737 | 509 | 13.62 | 1739 | 46.53 | 1,815 | 48.57 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 4 | 80.00 | 1 | 20.00 |
| 32 | Xiaomi | 313 | 95 | 30.35 | 300 | 95.85 | 13 | 4.15 |
| 33 | Zyxel | 1,317 | 125 | 9.49 | 663 | 50.34 | 570 | 43.28 |
| - | **Total** | 22,548 | 3,333 | 14.78 | 11,630 | 51.58 | 8,737 | 38.75 |

Table A.40: Pseudo Random Number Generators (PRNGs) cryptographic misuses (found from entry and possibly $\phi$).

| # | Vendor | # unpacked firmwares | K1 | % | K2 | % | K3 | % | K4 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 585 | 44.69 | 272 | 20.78 | 0 | 0.00 | 289 | 22.08 | 467 | 35.68 |
| 2 | AVM | 102 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 102 | 100.00 |
| 3 | Actiontec | 5 | 0 | 0.00 | 1 | 20.00 | 0 | 0.00 | 1 | 20.00 | 4 | 80.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 6 | 8.45 | 0 | 0.00 | 6 | 8.45 | 65 | 91.55 |
| 6 | Arris | 7 | 4 | 57.14 | 3 | 42.86 | 4 | 57.14 | 3 | 42.86 | 0 | 0.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 6 | 13.33 | 0 | 0.00 | 6 | 13.33 | 39 | 86.67 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | Buffalo | 4 | 0 | 0.00 | 1 | 25.00 | 0 | 0.00 | 1 | 25.00 | 3 | 75.00 |
| 9 | D-Link | 2,116 | 136 | 6.43 | 363 | 17.16 | 0 | 0.00 | 438 | 20.70 | 1,669 | 78.88 |
| 10 | Dell | 122 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 122 | 100.00 |
| 11 | DrayTek | 178 | 0 | 0.00 | 133 | 74.72 | 62 | 34.83 | 133 | 74.72 | 29 | 16.29 |
| 12 | EdiMax | 297 | 0 | 0.00 | 71 | 23.91 | 0 | 0.00 | 73 | 24.58 | 224 | 75.42 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 3 | 27.27 | 8 | 72.73 |
| 16 | LinkSys | 195 | 47 | 24.10 | 91 | 46.67 | 45 | 23.08 | 96 | 49.23 | 99 | 50.77 |
| 17 | MicroTik | 814 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 814 | 100.00 |
| 18 | NETGEAR | 8,061 | 28 | 0.35 | 475 | 5.89 | 2 | 0.02 | 4,611 | 57.20 | 3,450 | 42.80 |
| 19 | Netis | 114 | 0 | 0.00 | 98 | 85.96 | 0 | 0.00 | 98 | 85.96 | 16 | 14.04 |
| 20 | Planet | 418 | 0 | 0.00 | 78 | 18.66 | 0 | 0.00 | 82 | 19.62 | 336 | 80.38 |
| 21 | QNAP | 109 | 29 | 26.61 | 6 | 5.50 | 0 | 0.00 | 6 | 5.50 | 80 | 73.39 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 0 | 0.00 | 37 | 11.60 | 3 | 0.94 | 200 | 62.70 | 119 | 37.30 |
| 24 | TP-Link | 2,069 | 49 | 2.37 | 781 | 37.75 | 1 | 0.05 | 784 | 37.89 | 1,254 | 60.61 |
| 25 | Tenda | 367 | 0 | 0.00 | 52 | 14.17 | 4 | 1.09 | 52 | 14.17 | 311 | 84.74 |
| 26 | Tenvis | 7 | 0 | 0.00 | 2 | 28.57 | 0 | 0.00 | 2 | 28.57 | 5 | 71.43 |
| 27 | Thuraya | 2 | 0 | 0.00 | 2 | 100.00 | 0 | 0.00 | 2 | 100.00 | 0 | 0.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 3 | 2.08 | 0 | 0.00 | 5 | 3.47 | 139 | 96.53 |
| 29 | Trendnet | 267 | 19 | 7.12 | 93 | 34.83 | 16 | 5.99 | 95 | 35.58 | 172 | 64.42 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 78 | 2.09 | 76 | 2.03 | 426 | 11.40 | 3,235 | 86.57 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 | 0 | 0.00 |
| 32 | Xiaomi | 313 | 0 | 0.00 | 299 | 95.53 | 68 | 21.73 | 299 | 95.53 | 14 | 4.47 |
| 33 | Zyxel | 1,317 | 1 | 0.08 | 222 | 16.86 | 164 | 12.45 | 352 | 26.73 | 833 | 63.25 |
| - | **Total** | 22,548 | 898 | 3.98 | 3,173 | 14.07 | 445 | 1.97 | 8,068 | 35.78 | 13,632 | 60.46 |

Table A.41: Key Derivation Functions (KDFs) and Password Based Encryption (PBE) overall cryptographic misuses (found from entry and possibly $\phi$).

| # | Vendor | # unpacked firmwares | M1 | % | M2 | % | M3 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 0 | 0.00 | 183 | 13.98 | 29 | 2.22 | 1,126 | 86.02 |
| 2 | AVM | 102 | 0 | 0.00 | 5 | 4.90 | 0 | 0.00 | 97 | 95.10 |
| 3 | Actiontec | 5 | 0 | 0.00 | 2 | 40.00 | 0 | 0.00 | 3 | 60.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 33 | 46.48 | 8 | 11.27 | 38 | 53.52 |
| 6 | Arris | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 0 | 0.00 | 165 | 7.80 | 0 | 0.00 | 1,951 | 92.20 |
| 10 | Dell | 122 | 0 | 0.00 | 41 | 33.61 | 0 | 0.00 | 81 | 66.39 |
| 11 | DrayTek | 178 | 150 | 84.27 | 23 | 12.92 | 0 | 0.00 | 28 | 15.73 |
| 12 | EdiMax | 297 | 0 | 0.00 | 12 | 4.04 | 0 | 0.00 | 285 | 95.96 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 3 | 27.27 | 0 | 0.00 | 8 | 72.73 |
| 16 | LinkSys | 195 | 0 | 0.00 | 85 | 43.59 | 1 | 0.51 | 110 | 56.41 |
| 17 | MicroTik | 814 | 0 | 0.00 | 350 | 43.00 | 0 | 0.00 | 464 | 57.00 |
| 18 | NETGEAR | 8,061 | 182 | 2.26 | 69 | 0.86 | 32 | 0.40 | 7,793 | 96.68 |
| 19 | Netis | 114 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 114 | 100.00 |
| 20 | Planet | 418 | 0 | 0.00 | 58 | 13.88 | 0 | 0.00 | 360 | 86.12 |
| 21 | QNAP | 109 | 0 | 0.00 | 13 | 11.93 | 0 | 0.00 | 96 | 88.07 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 0 | 0.00 | 160 | 50.16 | 1 | 0.31 | 159 | 49.84 |
| 24 | TP-Link | 2,069 | 0 | 0.00 | 200 | 9.67 | 7 | 0.34 | 1,869 | 90.33 |
| 25 | Tenda | 367 | 0 | 0.00 | 23 | 6.27 | 1 | 0.27 | 344 | 93.73 |
| 26 | Tenvis | 7 | 0 | 0.00 | 1 | 14.29 | 0 | 0.00 | 6 | 85.71 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 4 | 2.78 | 1 | 0.69 | 139 | 96.53 |
| 29 | Trendnet | 267 | 0 | 0.00 | 13 | 4.87 | 0 | 0.00 | 254 | 95.13 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 308 | 8.24 | 0 | 0.00 | 3,429 | 91.76 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 4 | 80.00 | 0 | 0.00 | 1 | 20.00 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 32 | Xiaomi | 313 | 0 | 0.00 | 14 | 4.47 | 0 | 0.00 | 299 | 95.53 |
| 33 | Zyxel | 1317 | 0 | 0.00 | 31 | 2.35 | 0 | 0.00 | 1286 | 97.65 |
| - | **Total** | 22,548 | 332 | 1.47 | 1,800 | 7.98 | 80 | 0.35 | 20,421 | 90.57 |

Table A.42: Message Authentication Codes (MACs) overall cryptographic misuses (found from entry and possibly $\phi$).

| # | Vendor | # unpacked firmwares | A1 | % | A2 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 0 | 0.00 | 0 | 0.00 | 1,309 | 100.00 |
| 2 | AVM | 102 | 0 | 0.00 | 0 | 0.00 | 102 | 100.00 |
| 3 | Actiontec | 5 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 0 | 0.00 | 71 | 100.00 |
| 6 | Arris | 7 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 0 | 0.00 | 0 | 0.00 | 2,116 | 100.00 |
| 10 | Dell | 122 | 0 | 0.00 | 0 | 0.00 | 122 | 100.00 |
| 11 | DrayTek | 178 | 0 | 0.00 | 0 | 0.00 | 178 | 100.00 |
| 12 | EdiMax | 297 | 0 | 0.00 | 0 | 0.00 | 297 | 100.00 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 0 | 0.00 | 11 | 100.00 |
| 16 | LinkSys | 195 | 0 | 0.00 | 0 | 0.00 | 195 | 100.00 |
| 17 | MicroTik | 814 | 0 | 0.00 | 0 | 0.00 | 814 | 100.00 |
| 18 | NETGEAR | 8,061 | 0 | 0.00 | 0 | 0.00 | 8,061 | 100.00 |
| 19 | Netis | 114 | 0 | 0.00 | 0 | 0.00 | 114 | 100.00 |
| 20 | Planet | 418 | 0 | 0.00 | 0 | 0.00 | 418 | 100.00 |
| 21 | QNAP | 109 | 0 | 0.00 | 0 | 0.00 | 109 | 100.00 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 0 | 0.00 | 0 | 0.00 | 319 | 100.00 |
| 24 | TP-Link | 2,069 | 0 | 0.00 | 0 | 0.00 | 2,069 | 100.00 |
| 25 | Tenda | 367 | 0 | 0.00 | 0 | 0.00 | 367 | 100.00 |
| 26 | Tenvis | 7 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 0 | 0.00 | 144 | 100.00 |
| 29 | Trendnet | 267 | 0 | 0.00 | 0 | 0.00 | 267 | 100.00 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 0 | 0.00 | 3,737 | 100.00 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 32 | Xiaomi | 313 | 0 | 0.00 | 0 | 0.00 | 313 | 100.00 |
| 33 | Zyxel | 1,317 | 0 | 0.00 | 0 | 0.00 | 1,317 | 100.00 |
| - | **Total** | 22,548 | 0 | 0.00 | 0 | 0.00 | 22,548 | 100.00 |

Table A.43: Authenticated encryption/decryption and AEAD overall cryptographic misuses (found from entry and possible $\phi$).

| # | Vendor | # unpacked firmwares | S1 | % | S2 | % | S3 | % | S4 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 0 | 0.00 | 546 | 41.71 | 0 | 0.00 | 539 | 41.18 | 763 | 58.29 |
| 2 | AVM | 102 | 0 | 0.00 | 5 | 4.90 | 0 | 0.00 | 5 | 4.90 | 97 | 95.10 |
| 3 | Actiontec | 5 | 0 | 0.00 | 1 | 20.00 | 0 | 0.00 | 1 | 20.00 | 4 | 80.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 10 | 14.08 | 0 | 0.00 | 54 | 76.06 | 17 | 23.94 |
| 6 | Arris | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 203 | 9.59 | 356 | 16.82 | 9 | 0.43 | 219 | 10.35 | 1,587 | 75.00 |
| 10 | Dell | 122 | 0 | 0.00 | 41 | 33.61 | 0 | 0.00 | 41 | 33.61 | 81 | 66.39 |
| 11 | DrayTek | 178 | 83 | 46.63 | 5 | 2.81 | 0 | 0.00 | 9 | 5.06 | 91 | 51.12 |
| 12 | EdiMax | 297 | 0 | 0.00 | 4 | 1.35 | 0 | 0.00 | 4 | 1.35 | 293 | 98.65 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 3 | 27.27 | 0 | 0.00 | 3 | 27.27 | 8 | 72.73 |
| 16 | LinkSys | 195 | 2 | 1.03 | 86 | 44.10 | 0 | 0.00 | 99 | 50.77 | 94 | 48.21 |
| 17 | MicroTik | 814 | 0 | 0.00 | 190 | 23.34 | 0 | 0.00 | 0 | 0.00 | 624 | 76.66 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 18 | NETGEAR | 8,061 | 183 | 2.27 | 1,714 | 21.26 | 12 | 0.15 | 610 | 7.57 | 6,321 | 78.41 |
| 19 | Netis | 114 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 114 | 100.00 |
| 20 | Planet | 418 | 0 | 0.00 | 60 | 14.35 | 0 | 0.00 | 86 | 20.57 | 332 | 79.43 |
| 21 | QNAP | 109 | 10 | 9.17 | 23 | 21.10 | 0 | 0.00 | 23 | 21.10 | 86 | 78.90 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 0 | 0.00 | 165 | 51.72 | 0 | 0.00 | 196 | 61.44 | 123 | 38.56 |
| 24 | TP-Link | 2,069 | 70 | 3.38 | 162 | 7.83 | 0 | 0.00 | 137 | 6.62 | 1,854 | 89.61 |
| 25 | Tenda | 367 | 0 | 0.00 | 3 | 0.82 | 0 | 0.00 | 22 | 5.99 | 344 | 93.73 |
| 26 | Tenvis | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 4 | 2.78 | 0 | 0.00 | 4 | 2.78 | 140 | 97.22 |
| 29 | Trendnet | 267 | 0 | 0.00 | 34 | 12.73 | 0 | 0.00 | 29 | 10.86 | 228 | 85.39 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 259 | 6.93 | 0 | 0.00 | 387 | 10.36 | ,3350 | 89.64 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 4 | 80.00 | 0 | 0.00 | 4 | 80.00 | 1 | 20.00 |
| 32 | Xiaomi | 313 | 18 | 5.75 | 14 | 4.47 | 0 | 0.00 | 14 | 4.47 | 281 | 89.78 |
| 33 | Zyxel | 1,317 | 0 | 0.00 | 105 | 7.97 | 0 | 0.00 | 344 | 26.12 | 968 | 73.50 |
| - | **Total** | 22,548 | 569 | 2.52 | 3,794 | 16.83 | 21 | 0.09 | 2,830 | 12.55 | 17,889 | 79.34 |

Table A.44: Symmetric Key Cryptography overall cryptographic misuses (found from entry and not discovered $\phi$).

| # | Vendor | # unpacked firmwares | P1 | % | P2 | % | P3 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 347 | 26.51 | 0 | 0.00 | 1 | 0.08 | 961 | 73.41 |
| 2 | AVM | 102 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 102 | 100.00 |
| 3 | Actiontec | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 3 | 4.23 | 0 | 0.00 | 0 | 0.00 | 68 | 95.77 |
| 6 | Arris | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 25 | 1.18 | 0 | 0.00 | 167 | 7.89 | 1,924 | 90.93 |
| 10 | Dell | 122 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 122 | 100.00 |
| 11 | DrayTek | 178 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 178 | 100.00 |
| 12 | EdiMax | 297 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 297 | 100.00 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 11 | 100.00 |
| 16 | LinkSys | 195 | 1 | 0.51 | 0 | 0.00 | 2 | 1.03 | 192 | 98.46 |
| 17 | MicroTik | 814 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 814 | 100.00 |
| 18 | NETGEAR | 8,061 | 3197 | 39.66 | 0 | 0.00 | 1,109 | 13.76 | 4,618 | 57.29 |
| 19 | Netis | 114 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 114 | 100.00 |
| 20 | Planet | 418 | 0 | 0.00 | 0 | 0.00 | 24 | 5.74 | 394 | 94.26 |
| 21 | QNAP | 109 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 109 | 100.00 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 69 | 21.63 | 0 | 0.00 | 108 | 33.86 | 144 | 45.14 |
| 24 | TP-Link | 2,069 | 60 | 2.90 | 0 | 0.00 | 63 | 3.04 | 1,958 | 94.64 |
| 25 | Tenda | 367 | 3 | 0.82 | 0 | 0.00 | 0 | 0.00 | 364 | 99.18 |
| 26 | Tenvis | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 1 | 0.69 | 0 | 0.00 | 0 | 0.00 | 143 | 99.31 |
| 29 | Trendnet | 267 | 0 | 0.00 | 0 | 0.00 | 1 | 0.37 | 266 | 99.63 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 0 | 0.00 | 111 | 2.97 | 3,626 | 97.03 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 0 | 0.00 | 4 | 80.00 | 1 | 20.00 |
| 32 | Xiaomi | 313 | 96 | 30.67 | 0 | 0.00 | 14 | 4.47 | 203 | 64.86 |
| 33 | Zyxel | 1,317 | 48 | 3.64 | 0 | 0.00 | 0 | 0.00 | 1,269 | 96.36 |
| - | **Total** | 22,548 | 3,850 | 17.07 | 0 | 0.00 | 1,604 | 7.11 | 17,971 | 79.70 |

Table A.45: Public Key Cryptography overall cryptographic misuses (found from entry and not discovered $\phi$).

| # | Vendor | # unpacked firmwares | R1 | % | R2 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 0 | 0.00 | 908 | 69.37 | 401 | 30.63 |
| 2 | AVM | 102 | 0 | 0.00 | 12 | 11.76 | 90 | 88.24 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | Actiontec | 5 | 0 | 0.00 | 5 | 100.00 | 0 | 0.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 0 | 0.00 | 71 | 100.00 |
| 6 | Arris | 7 | 0 | 0.00 | 4 | 57.14 | 3 | 42.86 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 37 | 1.75 | 950 | 44.90 | 1,142 | 53.97 |
| 10 | Dell | 122 | 0 | 0.00 | 20 | 16.39 | 102 | 83.61 |
| 11 | DrayTek | 178 | 0 | 0.00 | 99 | 55.62 | 79 | 44.38 |
| 12 | EdiMax | 297 | 0 | 0.00 | 22 | 7.41 | 275 | 92.59 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 1 | 20.00 | 4 | 80.00 |
| 14 | HP | 17 | 0 | 0.00 | 5 | 29.41 | 12 | 70.59 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 5 | 45.45 | 6 | 54.55 |
| 16 | LinkSys | 195 | 6 | 3.08 | 117 | 60.00 | 76 | 38.97 |
| 17 | MicroTik | 814 | 0 | 0.00 | 632 | 77.64 | 182 | 22.36 |
| 18 | NETGEAR | 8,061 | 2140 | 26.55 | 4,372 | 54.24 | 1,882 | 23.35 |
| 19 | Netis | 114 | 0 | 0.00 | 7 | 6.14 | 107 | 93.86 |
| 20 | Planet | 418 | 4 | 0.96 | 149 | 35.65 | 269 | 64.35 |
| 21 | QNAP | 109 | 0 | 0.00 | 27 | 24.77 | 82 | 75.23 |
| 22 | Rotek | 1 | 0 | 0.00 | 1 | 100.00 | 0 | 0.00 |
| 23 | Synology | 319 | 46 | 14.42 | 186 | 58.31 | 133 | 41.69 |
| 24 | TP-Link | 2,069 | 54 | 2.61 | 1,202 | 58.10 | 858 | 41.47 |
| 25 | Tenda | 367 | 0 | 0.00 | 92 | 25.07 | 275 | 74.93 |
| 26 | Tenvis | 7 | 0 | 0.00 | 2 | 28.57 | 5 | 71.43 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 9 | 6.25 | 135 | 93.75 |
| 29 | Trendnet | 267 | 0 | 0.00 | 68 | 25.47 | 199 | 74.53 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 1,739 | 46.53 | 1,998 | 53.47 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 4 | 80.00 | 1 | 20.00 |
| 32 | Xiaomi | 313 | 95 | 30.35 | 300 | 95.85 | 13 | 4.15 |
| 33 | Zyxel | 1,317 | 109 | 8.28 | 663 | 50.34 | 570 | 43.28 |
| - | **Total** | 22,548 | 2,491 | 11.05 | 11,601 | 51.45 | 9,021 | 40.01 |

Table A.46: Pseudo Random Number Generators (PRNGs) cryptographic misuses (found from entry and not discovered $\phi$).

| # | Vendor | # unpacked firmwares | K1 | % | K2 | % | K3 | % | K4 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 6 | 0.46 | 8 | 0.61 | 0 | 0.00 | 8 | 0.61 | 1,300 | 99.31 |
| 2 | AVM | 102 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 102 | 100.00 |
| 3 | Actiontec | 5 | 0 | 0.00 | 1 | 20.00 | 0 | 0.00 | 1 | 20.00 | 4 | 80.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 71 | 100.00 |
| 6 | Arris | 7 | 4 | 57.14 | 3 | 42.86 | 4 | 57.14 | 3 | 42.86 | 0 | 0.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 1 | 2.22 | 0 | 0.00 | 1 | 2.22 | 44 | 97.78 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 133 | 6.29 | 167 | 7.89 | 0 | 0.00 | 226 | 10.68 | 1,878 | 88.75 |
| 10 | Dell | 122 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 122 | 100.00 |
| 11 | DrayTek | 178 | 0 | 0.00 | 0 | 0.00 | 62 | 34.83 | 0 | 0.00 | 116 | 65.17 |
| 12 | EdiMax | 297 | 0 | 0.00 | 14 | 4.71 | 0 | 0.00 | 14 | 4.71 | 283 | 95.29 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 3 | 27.27 | 8 | 72.73 |
| 16 | LinkSys | 195 | 47 | 24.10 | 49 | 25.13 | 45 | 23.08 | 49 | 25.13 | 146 | 74.87 |
| 17 | MicroTik | 814 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 814 | 100.00 |
| 18 | NETGEAR | 8,061 | 28 | 0.35 | 280 | 3.47 | 2 | 0.02 | 4,384 | 54.39 | 3,677 | 45.61 |
| 19 | Netis | 114 | 0 | 0.00 | 2 | 1.75 | 0 | 0.00 | 2 | 1.75 | 112 | 98.25 |
| 20 | Planet | 418 | 0 | 0.00 | 67 | 16.03 | 0 | 0.00 | 71 | 16.99 | 347 | 83.01 |
| 21 | QNAP | 109 | 29 | 26.61 | 6 | 5.50 | 0 | 0.00 | 6 | 5.50 | 80 | 73.39 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 0 | 0.00 | 37 | 11.60 | 3 | 0.94 | 200 | 62.70 | 119 | 37.30 |
| 24 | TP-Link | 2,069 | 49 | 2.37 | 719 | 34.75 | 1 | 0.05 | 719 | 34.75 | 1,319 | 63.75 |
| 25 | Tenda | 367 | 0 | 0.00 | 4 | 1.09 | 0 | 0.00 | 4 | 1.09 | 363 | 98.91 |
| 26 | Tenvis | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 27 | Thuraya | 2 | 0 | 0.00 | 2 | 100.00 | 0 | 0.00 | 2 | 100.00 | 0 | 0.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 1 | 0.69 | 0 | 0.00 | 1 | 0.69 | 143 | 99.31 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 29 | Trendnet | 267 | 19 | 7.12 | 43 | 16.10 | 16 | 5.99 | 43 | 16.10 | 224 | 83.90 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 78 | 2.09 | 76 | 2.03 | 426 | 11.40 | 3,235 | 86.57 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 4 | 80.00 | 1 | 20.00 |
| 32 | Xiaomi | 313 | 0 | 0.00 | 299 | 95.53 | 68 | 21.73 | 299 | 95.53 | 14 | 4.47 |
| 33 | Zyxel | 1,317 | 1 | 0.08 | 120 | 9.11 | 164 | 12.45 | 196 | 14.88 | 957 | 72.67 |
| - | **Total** | 22,548 | 316 | 1.40 | 1,901 | 8.43 | 441 | 1.96 | 6,662 | 29.55 | 15,513 | 68.80 |

Table A.47: Key Derivation Functions (KDFs) and Password Based Encryption (PBE) cryptographic misuses (found from entry and not discovered $\phi$).

| # | Vendor | # unpacked firmwares | M1 | % | M2 | % | M3 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 0 | 0.00 | 182 | 13.90 | 29 | 2.22 | 1,127 | 86.10 |
| 2 | AVM | 102 | 0 | 0.00 | 5 | 4.90 | 0 | 0.00 | 97 | 95.10 |
| 3 | Actiontec | 5 | 0 | 0.00 | 2 | 40.00 | 0 | 0.00 | 3 | 60.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 33 | 46.48 | 8 | 11.27 | 38 | 53.52 |
| 6 | Arris | 7 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 0 | 0.00 | 163 | 7.70 | 0 | 0.00 | 1,953 | 92.30 |
| 10 | Dell | 122 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 122 | 100.00 |
| 11 | DrayTek | 178 | 93 | 52.25 | 5 | 2.81 | 0 | 0.00 | 85 | 47.75 |
| 12 | EdiMax | 297 | 0 | 0.00 | 10 | 3.37 | 0 | 0.00 | 287 | 96.63 |
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 3 | 27.27 | 0 | 0.00 | 8 | 72.73 |
| 16 | LinkSys | 195 | 0 | 0.00 | 85 | 43.59 | 1 | 0.51 | 110 | 56.41 |
| 17 | MicroTik | 814 | 0 | 0.00 | 350 | 43.00 | 0 | 0.00 | 464 | 57.00 |
| 18 | NETGEAR | 8,061 | 182 | 2.26 | 69 | 0.86 | 32 | 0.40 | 7,793 | 96.68 |
| 19 | Netis | 114 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 114 | 100.00 |
| 20 | Planet | 418 | 0 | 0.00 | 57 | 13.64 | 0 | 0.00 | 361 | 86.36 |
| 21 | QNAP | 109 | 0 | 0.00 | 13 | 11.93 | 0 | 0.00 | 96 | 88.07 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 0 | 0.00 | 160 | 50.16 | 1 | 0.31 | 159 | 49.84 |
| 24 | TP-Link | 2,069 | 0 | 0.00 | 112 | 5.41 | 7 | 0.34 | 1,957 | 94.59 |
| 25 | Tenda | 367 | 0 | 0.00 | 23 | 6.27 | 1 | 0.27 | 344 | 93.73 |
| 26 | Tenvis | 7 | 0 | 0.00 | 1 | 14.29 | 0 | 0.00 | 6 | 85.71 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 4 | 2.78 | 0 | 0.00 | 140 | 97.22 |
| 29 | Trendnet | 267 | 0 | 0.00 | 12 | 4.49 | 0 | 0.00 | 255 | 95.51 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 308 | 8.24 | 0 | 0.00 | 3,429 | 91.76 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 4 | 80.00 | 0 | 0.00 | 1 | 20.00 |
| 32 | Xiaomi | 313 | 0 | 0.00 | 14 | 4.47 | 0 | 0.00 | 299 | 95.53 |
| 33 | Zyxel | 1,317 | 0 | 0.00 | 31 | 2.35 | 0 | 0.00 | 1,286 | 97.65 |
| - | **Total** | 22,548 | 275 | 1.22 | 1,646 | 7.30 | 79 | 0.35 | 20,615 | 91.43 |

Table A.48: Message Authentication Codes (MACs) overall cryptographic misuses (found from entry and not discovered $\phi$).

| # | Vendor | # unpacked firmwares | A1 | % | A2 | % | No violation | % |
|---|---|---|---|---|---|---|---|---|
| 1 | ASUS | 1,309 | 0 | 0.00 | 0 | 0.00 | 1,309 | 100.00 |
| 2 | AVM | 102 | 0 | 0.00 | 0 | 0.00 | 102 | 100.00 |
| 3 | Actiontec | 5 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 4 | Addvaluetech | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| 5 | Alfa | 71 | 0 | 0.00 | 0 | 0.00 | 71 | 100.00 |
| 6 | Arris | 7 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 7 | Belkin | 45 | 0 | 0.00 | 0 | 0.00 | 45 | 100.00 |
| 8 | Buffalo | 4 | 0 | 0.00 | 0 | 0.00 | 4 | 100.00 |
| 9 | D-Link | 2,116 | 0 | 0.00 | 0 | 0.00 | 2,116 | 100.00 |
| 10 | Dell | 122 | 0 | 0.00 | 0 | 0.00 | 122 | 100.00 |
| 11 | DrayTek | 178 | 0 | 0.00 | 0 | 0.00 | 178 | 100.00 |
| 12 | EdiMax | 297 | 0 | 0.00 | 0 | 0.00 | 297 | 100.00 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 13 | FOSCAM | 5 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 14 | HP | 17 | 0 | 0.00 | 0 | 0.00 | 17 | 100.00 |
| 15 | Inmarsat | 11 | 0 | 0.00 | 0 | 0.00 | 11 | 100.00 |
| 16 | LinkSys | 195 | 0 | 0.00 | 0 | 0.00 | 195 | 100.00 |
| 17 | MicroTik | 814 | 0 | 0.00 | 0 | 0.00 | 814 | 100.00 |
| 18 | NETGEAR | 8,061 | 0 | 0.00 | 0 | 0.00 | 8,061 | 100.00 |
| 19 | Netis | 114 | 0 | 0.00 | 0 | 0.00 | 114 | 100.00 |
| 20 | Planet | 418 | 0 | 0.00 | 0 | 0.00 | 418 | 100.00 |
| 21 | QNAP | 109 | 0 | 0.00 | 0 | 0.00 | 109 | 100.00 |
| 22 | Rotek | 1 | 0 | 0.00 | 0 | 0.00 | 1 | 100.00 |
| 23 | Synology | 319 | 0 | 0.00 | 0 | 0.00 | 319 | 100.00 |
| 24 | TP-Link | 2,069 | 0 | 0.00 | 0 | 0.00 | 2,069 | 100.00 |
| 25 | Tenda | 367 | 0 | 0.00 | 0 | 0.00 | 367 | 100.00 |
| 26 | Tenvis | 7 | 0 | 0.00 | 0 | 0.00 | 7 | 100.00 |
| 27 | Thuraya | 2 | 0 | 0.00 | 0 | 0.00 | 2 | 100.00 |
| 28 | Totolink | 144 | 0 | 0.00 | 0 | 0.00 | 144 | 100.00 |
| 29 | Trendnet | 267 | 0 | 0.00 | 0 | 0.00 | 267 | 100.00 |
| 30 | Ubiquiti | 3,737 | 0 | 0.00 | 0 | 0.00 | 3,737 | 100.00 |
| 31 | Western-Digital | 5 | 0 | 0.00 | 0 | 0.00 | 5 | 100.00 |
| 32 | Xiaomi | 313 | 0 | 0.00 | 0 | 0.00 | 313 | 100.00 |
| 33 | Zyxel | 1,317 | 0 | 0.00 | 0 | 0.00 | 1,317 | 100.00 |
| - | **Total** | 22,548 | 0 | 0.00 | 0 | 0.00 | 22,548 | 100.00 |

Table A.49: Authenticated encryption/decryption and AEAD overall cryptographic misuses (found from entry and not discovered $\phi$).

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| ***S1*** - Constant Encryption/Decryption Keys | | | ***S2*** - Usage of ECB mode of operation | | |
| D-Link | ’imgdecrypt’ | 134 | ASUS | ’afppasswd’ | 403 |
| D-Link | ’smm’ | 69 | ASUS | ’cfg_server’ | 367 |
| D-Link | ’firebase’ | 69 | ASUS | ’cfg_client’ | 357 |
| D-Link | ’protest’ | 9 | ASUS | ’wpa_supplicant’ | 152 |
| DrayTek | ’mainfunction.cgi’ | 44 | ASUS | ’hostapd’ | 96 |
| DrayTek | ’goahead’ | 34 | ASUS | ’wpa_supplicant-2.7’ | 29 |
| DrayTek | ’oneTimeCall’ | 30 | ASUS | ’chilli’ | 12 |
| DrayTek | ’dray_apm’ | 16 | ASUS | ’chilli_response’ | 5 |
| DrayTek | ’dray_fwup’ | 13 | ASUS | ’bluetoothd’ | 5 |
| DrayTek | ’tr069_client’ | 13 | ASUS | ’rc’ | 1 |
| LinkSys | ’main_bin’ | 2 | AVM | ’hostapd’ | 5 |
| NETGEAR | ’Netgear_ddns’ | 144 | AVM | ’wpa_supplicant’ | 5 |
| NETGEAR | ’firebase’ | 27 | Actiontec | ’stunnel’ | 1 |
| NETGEAR | ’smm’ | 27 | Alfa | ’wpad’ | 8 |
| NETGEAR | ’NetReadyAgent’ | 12 | Alfa | ’wpa_supplicant’ | 2 |
| QNAP | ’nasutil’ | 10 | D-Link | ’signalc’ | 117 |
| TP-Link | ’tdpServer’ | 62 | D-Link | ’hostapd’ | 74 |
| TP-Link | ’tdpd’ | 6 | D-Link | ’smm’ | 69 |
| TP-Link | ’dropbearmulti’ | 3 | D-Link | ’firebase’ | 69 |
| TP-Link | ’test_libgdpr’ | 2 | D-Link | ’wpa_supplicant’ | 60 |
| Xiaomi | ’securitypage’ | 18 | D-Link | ’stunnel’ | 32 |
| ***S3*** - Constant IV for various modes of operation | | | D-Link | ’linkd.out’ | 14 |
| D-Link | ’protest’ | 9 | D-Link | ’l7-feature’ | 8 |
| NETGEAR | ’NetReadyAgent’ | 12 | D-Link | ’ptcore’ | 7 |
| ***S4*** - Usage of ‘weak’ ciphers for encryption | | | D-Link | ’xsupplicant’ | 4 |
| ASUS | ’afppasswd’ | 403 | D-Link | ’elephantdrive’ | 2 |
| ASUS | ’hostapd’ | 96 | Dell | ’stunnel4’ | 41 |
| ASUS | ’wpa_supplicant-2.7’ | 29 | DrayTek | ’hostapd’ | 5 |
| ASUS | ’chilli’ | 12 | DrayTek | ’wpa_supplicant’ | 5 |
| ASUS | ’chilli_response’ | 5 | EdiMax | ’hostapd’ | 3 |
| AVM | ’hostapd’ | 5 | EdiMax | ’wpa_supplicant’ | 2 |
| AVM | ’wpa_supplicant’ | 5 | EdiMax | ’stunnel’ | 1 |
| Actiontec | ’stunnel’ | 1 | Inmarsat | ’hostapd’ | 3 |
| Alfa | ’wpad’ | 33 | LinkSys | ’stunnel’ | 77 |
| Alfa | ’wpa_supplicant’ | 21 | LinkSys | ’hostapd’ | 20 |
| D-Link | ’snmpd’ | 121 | LinkSys | ’wpa_supplicant’ | 18 |

| | | | | | |
|---|---|---|---|---|---|
| D-Link | 'snmptrap' | 120 | LinkSys | 'main_bin' | 2 |
| D-Link | 'hostapd' | 74 | MicroTik | 'ipsec' | 125 |
| D-Link | 'wpa_supplicant' | 60 | MicroTik | 'racoon' | 65 |
| D-Link | 'stunnel' | 32 | NETGEAR | 'readyNASVault' | 1038 |
| D-Link | 'ptcore' | 7 | NETGEAR | 'afppasswd' | 441 |
| D-Link | 'xsupplicant' | 4 | NETGEAR | 'fbwifi' | 208 |
| D-Link | 'snmpwalk' | 2 | NETGEAR | 'hostapd_app' | 101 |
| D-Link | 'snmpset' | 2 | NETGEAR | 'wpa_supplicant' | 83 |
| D-Link | 'snmpget' | 2 | NETGEAR | 'hostapd' | 67 |
| Dell | 'stunnel4' | 41 | NETGEAR | 'upAgent' | 54 |
| DrayTek | 'wpa_supplicant' | 9 | NETGEAR | 'firebase' | 27 |
| DrayTek | 'hostapd' | 4 | NETGEAR | 'smm' | 27 |
| EdiMax | 'hostapd' | 3 | NETGEAR | 'shttpd' | 11 |
| EdiMax | 'wpa_supplicant' | 2 | NETGEAR | 'funjsq_cli' | 10 |
| EdiMax | 'stunnel' | 1 | NETGEAR | 'stunnel' | 3 |
| Inmarsat | 'hostapd' | 3 | NETGEAR | 'lc_up' | 1 |
| LinkSys | 'stunnel' | 77 | NETGEAR | 'mongoose' | 1 |
| LinkSys | 'hostapd' | 36 | NETGEAR | 'wpa_supplicant-macsec' | 1 |
| LinkSys | 'wpa_supplicant' | 18 | Planet | 'wpa_supplicant' | 61 |
| NETGEAR | 'afppasswd' | 415 | Planet | 'stunnel' | 3 |
| NETGEAR | 'fbwifi' | 208 | QNAP | 'stunnel' | 23 |
| NETGEAR | 'hostapd_app' | 101 | Synology | 'wpa_supplicant' | 165 |
| NETGEAR | 'wpa_supplicant' | 83 | Synology | 'hostapd' | 144 |
| NETGEAR | 'hostapd' | 60 | Synology | 'img_backup' | 10 |
| NETGEAR | 'upAgent' | 54 | Synology | 'img_restore' | 6 |
| NETGEAR | 'funjsq_cli' | 10 | Synology | 'detect_monitor' | 6 |
| NETGEAR | 'stunnel' | 3 | Synology | 'synoimgbkptool' | 6 |
| NETGEAR | 'dbcfg_export' | 1 | Synology | 'synoappexport' | 4 |
| NETGEAR | 'wpa_supplicant-macsec' | 1 | Synology | 'img_worker' | 2 |
| NETGEAR | 'eapol_test' | 1 | Synology | 'synohbkpvfs' | 2 |
| NETGEAR | 'snmptrap' | 1 | TP-Link | 'wpa_supplicant' | 86 |
| NETGEAR | 'snmpd' | 1 | TP-Link | 'hostapd' | 85 |
| Planet | 'wpa_supplicant' | 61 | TP-Link | 'afppasswd' | 37 |
| Planet | 'snmpd' | 50 | TP-Link | 'mbedtls_aes-128-ecb' | 1 |
| Planet | 'stunnel' | 3 | TP-Link | 'mysqld' | 1 |
| Planet | 'snmptrap' | 1 | TP-Link | 'mariabackup' | 1 |
| QNAP | 'stunnel' | 23 | Tenda | 'hostapd' | 2 |
| QNAP | 'wpa_supplicant' | 13 | Tenda | 'wpa_supplicant' | 1 |
| Synology | 'rsync' | 196 | Tenda | 'udhcpd' | 1 |
| Synology | 'wpa_supplicant' | 183 | Totolink | 'wpa_supplicant' | 4 |
| Synology | 'hostapd' | 166 | Trendnet | 'wpa_supplicant' | 20 |
| TP-Link | 'wpa_supplicant' | 86 | Trendnet | 'daemon_fsp_app' | 16 |
| TP-Link | 'hostapd' | 77 | Trendnet | '0' | 6 |
| TP-Link | 'afppasswd' | 7 | Trendnet | 'stunnel' | 4 |
| Tenda | 'racoon' | 21 | Trendnet | 'hostapd' | 2 |
| Tenda | 'hostapd' | 2 | Ubiquiti | 'wpad' | 187 |
| Tenda | 'wpa_supplicant' | 1 | Ubiquiti | 'hostapd' | 61 |
| Totolink | 'wpa_supplicant' | 4 | Ubiquiti | 'wpa_supplicant' | 39 |
| Trendnet | 'wpa_supplicant' | 20 | Western-Digital | 'wpa_supplicant' | 4 |
| Trendnet | 'snmpd' | 7 | Western-Digital | 'hostapd' | 3 |
| Trendnet | '0' | 6 | Western-Digital | 'ntfsdecrypt' | 1 |
| Trendnet | 'hostapd' | 4 | Xiaomi | 'wpa_supplicant' | 13 |
| Trendnet | 'stunnel' | 4 | Xiaomi | '245506E' | 1 |
| Trendnet | 'mgntd' | 3 | Zyxel | 'stunnel' | 54 |
| Trendnet | 'ZNMPClient' | 2 | Zyxel | 'hostapd' | 41 |
| Ubiquiti | 'wpad' | 264 | Zyxel | 'wpa_supplicant' | 34 |
| Ubiquiti | 'snmpd' | 102 | Zyxel | 'httpd' | 5 |
| Ubiquiti | 'hostapd' | 67 | Zyxel | 'hostapd_0_8_x' | 1 |
| Ubiquiti | 'wpa_supplicant' | 53 | Zyxel | 'wpa_supplicant-macsec' | 1 |
| Western-Digital | 'wpa_supplicant' | 4 | - | - | - |
| Western-Digital | 'hostapd' | 3 | - | - | - |
| Xiaomi | 'wpa_supplicant' | 13 | - | - | - |
| Xiaomi | '245506E' | 1 | - | - | - |
| Zyxel | 'wpa_supplicant' | 285 | - | - | - |
| Zyxel | 'stunnel' | 54 | - | - | - |
| Zyxel | 'hostapd' | 41 | - | - | - |
| Zyxel | 'snmpd' | 5 | - | - | - |

| | | | | | |
|---|---|---|---|---|---|
| Zyxel | ’hostapd_0_8_x’ | 1 | - | - | - |
| Zyxel | ’wpa_supplicant-macsec’ | 1 | - | - | - |

Table A.50: Violated Binaries discovered for Symmetric Key Cryptography rules **S1**, **S2**, **S3** and **S4**

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| ***P1*** - Usage of insecure RSA encryption padding schemes | | | ***P3*** - X.509 certificate usage of ‘weak’ digest function | | |
| ASUS | ’cfg_client’ | 335 | ASUS | ’qmi_ip’ | 1 |
| ASUS | ’cfg_server’ | 14 | D-Link | ’mpop’ | 85 |
| Alfa | ’tor’ | 2 | D-Link | ’x509SelfSign’ | 75 |
| Alfa | ’rsa_test’ | 1 | D-Link | ’mapd’ | 28 |
| D-Link | ’captival_portal’ | 19 | D-Link | ’gencert’ | 5 |
| D-Link | ’httpd’ | 5 | D-Link | ’imspector’ | 2 |
| D-Link | ’shareport’ | 4 | LinkSys | ’mapd’ | 2 |
| D-Link | ’spt’ | 4 | NETGEAR | ’ntfsdecrypt’ | 860 |
| D-Link | ’EmbedThunderManager’ | 1 | NETGEAR | ’certgen’ | 222 |
| LinkSys | ’iperf3’ | 1 | NETGEAR | ’x509SelfSign’ | 27 |
| NETGEAR | ’fvdropbox’ | 2244 | Planet | ’monit’ | 24 |
| NETGEAR | ’avdu’ | 2014 | Synology | ’lftp’ | 88 |
| NETGEAR | ’fvamazon’ | 1232 | Synology | ’nzbget’ | 18 |
| NETGEAR | ’readynasd’ | 928 | Synology | ’ncat’ | 2 |
| NETGEAR | ’NetReadyAgent’ | 12 | TP-Link | ’httpd’ | 47 |
| NETGEAR | ’tincd’ | 10 | TP-Link | ’sslselfsign’ | 15 |
| NETGEAR | ’TPMFactoryUpd’ | 8 | TP-Link | ’mysqlimport’ | 1 |
| NETGEAR | ’dimclient’ | 4 | TP-Link | ’mysqlcheck’ | 1 |
| NETGEAR | ’iperf3’ | 1 | TP-Link | ’mysqlshow’ | 1 |
| Synology | ’synolicense_uninstall’ | 52 | TP-Link | ’mysqldump’ | 1 |
| Synology | ’sftpd’ | 17 | TP-Link | ’mysqlslap’ | 1 |
| Synology | ’synoddsm-hostd’ | 3 | TP-Link | ’mysqladmin’ | 1 |
| TP-Link | ’httpd’ | 36 | TP-Link | ’mysql’ | 1 |
| TP-Link | ’eap-mesh’ | 12 | TP-Link | ’mysqlbinlog’ | 1 |
| TP-Link | ’eapcs’ | 8 | TP-Link | ’mysqltest’ | 1 |
| TP-Link | ’o_p_test’ | 4 | Trendnet | ’ipheth-pair’ | 1 |
| TP-Link | ’rsa_decrypt’ | 4 | Ubiquiti | ’monit’ | 108 |
| TP-Link | ’tdpServer’ | 3 | Ubiquiti | ’snmpd’ | 20 |
| Tenda | ’eventdispatcher’ | 2 | Ubiquiti | ’httping’ | 3 |
| Tenda | ’racoon’ | 1 | Western-Digital | ’monit’ | 4 |
| Totolink | ’tincd’ | 1 | Western-Digital | ’ncat’ | 3 |
| Xiaomi | ’etm’ | 95 | Western-Digital | ’ntfsdecrypt’ | 1 |
| Xiaomi | ’rsa_test’ | 1 | Xiaomi | ’syslog-ng’ | 14 |
| Zyxel | ’zhttpd’ | 30 | - | - | - |
| Zyxel | ’zyxel_xmpp_client’ | 12 | - | - | - |
| Zyxel | ’zyxel_encrypt_hash’ | 9 | - | - | - |
| Zyxel | ’httpd’ | 5 | - | - | - |

Table A.51: Violated Binaries discovered for Public Key Cryptography rules **P1** and **P3**, entry and possible $\phi$ case.

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| ***R1*** - PRNG static seed | | | | | |
| ASUS | ’zebra’ | 137 | TP-Link | ’sessmngr’ | 6 |
| Actiontec | ’uhttpd’ | 1 | TP-Link | ’capwap’ | 3 |
| Buffalo | ’embeddd’ | 2 | Trendnet | ’zebra’ | 2 |
| D-Link | ’htpasswd’ | 36 | Ubiquiti | ’switchdrvr’ | 252 |
| D-Link | ’zebra’ | 23 | Ubiquiti | ’imi’ | 220 |
| D-Link | ’sys_commander.x’ | 2 | Ubiquiti | ’ripngd’ | 220 |
| D-Link | ’EmbedThunderManager’ | 1 | Ubiquiti | ’nsm’ | 220 |
| LinkSys | ’zebra’ | 23 | Ubiquiti | ’rsvpd’ | 220 |
| LinkSys | ’udhcpd’ | 4 | Ubiquiti | ’oamd’ | 220 |
| LinkSys | ’Mercury.snos’ | 2 | Ubiquiti | ’ldpd’ | 220 |
| LinkSys | ’init_nvram’ | 1 | Ubiquiti | ’ribd’ | 220 |
| NETGEAR | ’htpasswd’ | 2261 | Ubiquiti | ’ospfd’ | 220 |
| NETGEAR | ’zebra’ | 91 | Ubiquiti | ’ospf6d’ | 220 |
| NETGEAR | ’tincd’ | 10 | Ubiquiti | ’bgpd’ | 185 |

| | | | | | |
|---|---|---|---|---|---|
| NETGEAR | 'iss.exe' | 1 | Ubiquiti | 'zebra' | 37 |
| NETGEAR | 'portal' | 1 | Xiaomi | 'etm' | 95 |
| Planet | 'htpasswd' | 4 | Zyxel | 'login.cgi' | 84 |
| Synology | 'htpasswd' | 50 | Zyxel | 'dispatcher.cgi' | 36 |
| Synology | 'postgres' | 2 | Zyxel | 'htpasswd' | 25 |
| TP-Link | 'cet' | 45 | Zyxel | '433DEB' | 18 |
| TP-Link | 'msg_push' | 19 | Zyxel | 'zebra' | 16 |
| TP-Link | 'dhcpc' | 13 | Zyxel | '443F0F' | 9 |
| TP-Link | 'dhcpd' | 11 | Zyxel | '4729B2' | 9 |
| TP-Link | 'iked' | 10 | Zyxel | '432C64' | 9 |
| TP-Link | 'aaa' | 6 | Zyxel | '472073' | 3 |
| ***R2*** - Low entropy sources for seeds | | | | | |
| ASUS | 'aaews' | 616 | Synology | 'usbcopy-hook' | 112 |
| ASUS | 'mastiff' | 510 | Synology | 'usb-copy-notifier' | 112 |
| ASUS | 'watchquagga' | 436 | Synology | 'usb-copy-starter' | 110 |
| ASUS | 'cfg_server' | 367 | Synology | 'zip' | 45 |
| ASUS | 'httpd' | 195 | Synology | 'postgres' | 38 |
| ASUS | 'zebra' | 190 | Synology | 'network.cgi' | 36 |
| ASUS | 'miniupnpd' | 154 | Synology | 'upgrade.cgi' | 35 |
| ASUS | 'rc' | 51 | Synology | 'thumbnail.cgi' | 35 |
| ASUS | 'boa' | 16 | Synology | 'postgres32' | 34 |
| ASUS | 'cfg_client' | 15 | Synology | 'fileindexd' | 31 |
| ASUS | 'btgatt-server' | 8 | Synology | 'imap-login' | 28 |
| ASUS | 'bluealsa' | 7 | Synology | 'dovecot-auth' | 28 |
| ASUS | 'sip_proxy' | 3 | Synology | 'ssl-build-param' | 28 |
| ASUS | 'tr69c' | 2 | Synology | 'dovecot' | 28 |
| ASUS | 'newusers' | 2 | Synology | 'pop3-login' | 28 |
| AVM | 'mount.davfs' | 11 | Synology | 'postlock' | 27 |
| AVM | 'cloudmsgd' | 10 | Synology | 'pop3' | 27 |
| AVM | 'wlmngr2' | 2 | Synology | 'imap' | 27 |
| AVM | 'tr69c' | 1 | Synology | 'synodisk' | 24 |
| Actiontec | 'zebra' | 3 | Synology | 'scemd' | 15 |
| Actiontec | 'tr69c' | 2 | Synology | 'synobox' | 14 |
| Actiontec | 'uhttpd' | 1 | Synology | 'image_thumb.cgi' | 8 |
| Actiontec | 'sntp' | 1 | Synology | 'heartbeatd' | 6 |
| Actiontec | 'detectWANService' | 1 | Synology | 'synoswitchvlantool' | 5 |
| Actiontec | 'cm_logic' | 1 | Synology | 'synowolagentd' | 5 |
| Arris | 'ripngd' | 4 | Synology | 'git-fast-import' | 4 |
| D-Link | 'pppdo' | 183 | Synology | 'git-http-push' | 4 |
| D-Link | 'upnpc-ddns' | 173 | Synology | 'git-credential-store' | 4 |
| D-Link | 'vipsecureConfig' | 143 | Synology | 'git-remote-https' | 4 |
| D-Link | 'prog-cgi' | 126 | Synology | 'git-http-fetch' | 4 |
| D-Link | 'mpop' | 85 | Synology | 'synodbudd' | 4 |
| D-Link | 'admin.cgi' | 67 | Synology | 'dhclient' | 4 |
| D-Link | 'mailsend' | 63 | Synology | 'aa_cmd' | 3 |
| D-Link | 'da_adaptor' | 61 | Synology | 'iscsiadm' | 3 |
| D-Link | 'newp2p' | 59 | Synology | 'PkgSynoMan.cgi' | 3 |
| D-Link | 'snmpd' | 49 | Synology | 'synodatacollectd' | 2 |
| D-Link | 'httpd' | 46 | Synology | 'winbindd' | 2 |
| D-Link | 'cgibin' | 45 | Synology | 'nmblookup' | 2 |
| D-Link | 'mapd' | 45 | Synology | 'cloud-cleand' | 2 |
| D-Link | 'p2p_server' | 43 | Synology | 'cloud-control' | 2 |
| D-Link | 'shgw_watchdogd' | 41 | Synology | 'img_backup' | 2 |
| D-Link | 'x509SelfSign' | 34 | Synology | 'syno-cloud-syncd' | 2 |
| D-Link | 'zebra' | 32 | Synology | 'debug' | 2 |
| D-Link | 'afpd' | 26 | Synology | 'synologyfilemanager-authd' | 2 |
| D-Link | 'lighttpd' | 20 | Synology | 'pgbouncer' | 2 |
| D-Link | 'perl' | 20 | Synology | 'dig' | 2 |
| D-Link | 'jjhttpd' | 18 | Synology | 'CSTNVolChange' | 2 |
| D-Link | 'crtmpserver' | 18 | Synology | 'RestoreNode' | 2 |
| D-Link | 'linkd.out' | 14 | Synology | 'cloud-sync-encrypt-tool' | 2 |
| D-Link | 'test_ap' | 14 | Synology | 'cloud-cached' | 2 |
| D-Link | 'dv8_agent' | 13 | Synology | 'syno-cloud-clientd' | 2 |
| D-Link | 'op_server' | 13 | Synology | 'feasibility-check' | 2 |
| D-Link | 'mt-daapd' | 11 | Synology | 'cloud-sync-starter' | 2 |
| D-Link | 'hd_verify' | 11 | Synology | 'db-check' | 2 |

| | | | | | |
|---|---|---|---|---|---|
| D-Link | ’ipca’ | 10 | Synology | ’cloud-authd’ | 2 |
| D-Link | ’tr69c’ | 9 | Synology | ’syno-letsencrypt’ | 2 |
| D-Link | ’miniupnpd’ | 8 | Synology | ’synoupgrade’ | 1 |
| D-Link | ’commander’ | 8 | Synology | ’dms’ | 1 |
| D-Link | ’winbindd’ | 6 | Synology | ’main.cgi’ | 1 |
| D-Link | ’net’ | 6 | Synology | ’synotifyd’ | 1 |
| D-Link | ’onvifServer’ | 6 | Synology | ’ha.cgi’ | 1 |
| D-Link | ’GBhandler’ | 5 | Synology | ’virtual’ | 1 |
| D-Link | ’prog.cgi’ | 4 | Synology | ’local’ | 1 |
| D-Link | ’ripngd’ | 4 | Synology | ’bounce’ | 1 |
| D-Link | ’tr69’ | 4 | TP-Link | ’cloud-brd’ | 628 |
| D-Link | ’sudo’ | 4 | TP-Link | ’cloud-client’ | 361 |
| D-Link | ’webs’ | 4 | TP-Link | ’cet’ | 229 |
| D-Link | ’record_server’ | 3 | TP-Link | ’uac’ | 200 |
| D-Link | ’MAIL.VideoServer.strip’ | 3 | TP-Link | ’relayd’ | 174 |
| D-Link | ’watchquagga’ | 2 | TP-Link | ’miniupnpd’ | 144 |
| D-Link | ’ppp’ | 2 | TP-Link | ’cwmp’ | 109 |
| D-Link | ’dnsproxy’ | 2 | TP-Link | ’newusers’ | 80 |
| D-Link | ’vvctl’ | 2 | TP-Link | ’pure-pw’ | 67 |
| D-Link | ’newgrp’ | 2 | TP-Link | ’streamd’ | 58 |
| D-Link | ’resident’ | 2 | TP-Link | ’ipcamera’ | 58 |
| D-Link | ’hapClient’ | 2 | TP-Link | ’uhttpd’ | 40 |
| D-Link | ’agent’ | 1 | TP-Link | ’dsd’ | 34 |
| D-Link | ’accessctl’ | 1 | TP-Link | ’nvid’ | 32 |
| D-Link | ’lprm’ | 1 | TP-Link | ’nvrcore’ | 20 |
| D-Link | ’lpr’ | 1 | TP-Link | ’onboarding’ | 19 |
| D-Link | ’lpq’ | 1 | TP-Link | ’speaker’ | 19 |
| D-Link | ’lpd’ | 1 | TP-Link | ’storage’ | 14 |
| D-Link | ’lpc’ | 1 | TP-Link | ’cloud-sdk’ | 12 |
| D-Link | ’EmbedThunderManager’ | 1 | TP-Link | ’cloud_brd’ | 10 |
| D-Link | ’tr069’ | 1 | TP-Link | ’dcd’ | 8 |
| D-Link | ’tssa’ | 1 | TP-Link | ’eapcs’ | 8 |
| Dell | ’compmanager’ | 20 | TP-Link | ’httpd’ | 7 |
| DrayTek | ’mainfunction.cgi’ | 63 | TP-Link | ’predictd’ | 7 |
| DrayTek | ’onvif_func’ | 55 | TP-Link | ’v6plus’ | 6 |
| DrayTek | ’lighttpd’ | 46 | TP-Link | ’eap-cs’ | 5 |
| DrayTek | ’oneTimeCall’ | 30 | TP-Link | ’vod’ | 5 |
| DrayTek | ’dray_apm’ | 16 | TP-Link | ’voip_client’ | 4 |
| DrayTek | ’dhcrelay’ | 14 | TP-Link | ’zavim’ | 4 |
| DrayTek | ’acs’ | 9 | TP-Link | ’cloud_client’ | 3 |
| DrayTek | ’goahead’ | 2 | TP-Link | ’tr69c’ | 3 |
| EdiMax | ’zebra’ | 14 | TP-Link | ’zebra’ | 3 |
| EdiMax | ’boa’ | 4 | TP-Link | ’dig’ | 3 |
| EdiMax | ’btget’ | 3 | TP-Link | ’host’ | 3 |
| EdiMax | ’tr69c’ | 3 | TP-Link | ’avirasentinelfull’ | 3 |
| EdiMax | ’lighttpd’ | 2 | TP-Link | ’avirasentinellite’ | 3 |
| EdiMax | ’mailsend’ | 2 | TP-Link | ’aviraserviceselector’ | 3 |
| FOSCAM | ’jco_server’ | 1 | TP-Link | ’avirawatchdog’ | 3 |
| HP | ’lighttpd’ | 5 | TP-Link | ’aria2c’ | 3 |
| Inmarsat | ’ogg123’ | 3 | TP-Link | ’wlan-manager’ | 3 |
| Inmarsat | ’asterisk’ | 1 | TP-Link | ’mobile’ | 2 |
| Inmarsat | ’lighttpd’ | 1 | TP-Link | ’appcmd’ | 2 |
| LinkSys | ’dhclient’ | 86 | TP-Link | ’samba_multicall’ | 1 |
| LinkSys | ’httpd’ | 15 | TP-Link | ’mysqlslap’ | 1 |
| LinkSys | ’ripngd’ | 14 | TP-Link | ’mediaServer’ | 1 |
| LinkSys | ’zebra’ | 8 | Tenda | ’pppdForPptp’ | 33 |
| LinkSys | ’tr69c’ | 6 | Tenda | ’xl2tpdpppd’ | 33 |
| LinkSys | ’fwupd’ | 6 | Tenda | ’tr69c’ | 29 |
| LinkSys | ’dnsproxy’ | 4 | Tenda | ’httpd’ | 16 |
| LinkSys | ’cwmpCPE’ | 3 | Tenda | ’portal’ | 10 |
| LinkSys | ’bgpd’ | 3 | Tenda | ’zebra’ | 6 |
| LinkSys | ’lrhkprvsn’ | 3 | Tenda | ’pppdForPppServer’ | 5 |
| LinkSys | ’admin.cgi’ | 2 | Tenda | ’pppoa’ | 3 |
| LinkSys | ’ospfd’ | 2 | Tenda | ’ripngd’ | 2 |
| LinkSys | ’ospf6d’ | 2 | Tenda | ’bi’ | 2 |
| LinkSys | ’mailsend’ | 2 | Tenda | ’pppd244’ | 2 |
| LinkSys | ’boa’ | 1 | Tenda | ’pppd_3g’ | 1 |

| | | | | | |
|---|---|---|---|---|---|
| LinkSys | 'setup.cgi' | 1 | Tenda | 'pppd_245' | 1 |
| LinkSys | 'watchquagga' | 1 | Tenda | 'wlmngr2' | 1 |
| LinkSys | 'lighttpd' | 1 | Tenda | 'pppds' | 1 |
| LinkSys | 'LiveviewControlServer' | 1 | Tenvis | 'tutk' | 2 |
| MicroTik | 'ddns' | 580 | Totolink | 'ss-orig-redir' | 6 |
| MicroTik | 'ipsec' | 513 | Totolink | 'ss-orig-tunnel' | 6 |
| NETGEAR | 'readyNASVault' | 1790 | Totolink | 'ss-orig-local' | 6 |
| NETGEAR | 'mysqlmanager' | 714 | Totolink | 'ssr-redir' | 6 |
| NETGEAR | 'htdbm' | 714 | Totolink | 'ssr-local' | 6 |
| NETGEAR | 'httpd' | 581 | Totolink | 'tinc' | 5 |
| NETGEAR | 'zebra' | 382 | Totolink | 'ppp2d' | 2 |
| NETGEAR | 'auditd' | 289 | Totolink | 'rc' | 1 |
| NETGEAR | 'rcagentd' | 284 | Totolink | 'miniupnpd' | 1 |
| NETGEAR | 'rc_apps' | 185 | Trendnet | 'zebra' | 24 |
| NETGEAR | 'Netgear_ddns' | 144 | Trendnet | 'mailsend' | 10 |
| NETGEAR | 'ripngd' | 135 | Trendnet | 'snmpd' | 10 |
| NETGEAR | 'uhttpd' | 131 | Trendnet | 'lighttpd' | 10 |
| NETGEAR | 'mini_httpd' | 123 | Trendnet | 'vcm_serv' | 10 |
| NETGEAR | 'miniupnpd' | 119 | Trendnet | 'hicore' | 6 |
| NETGEAR | 'lighttpd' | 116 | Trendnet | 'webproc' | 4 |
| NETGEAR | 'nlogin.cgi' | 96 | Trendnet | 'pppds' | 4 |
| NETGEAR | 'apcomm' | 96 | Trendnet | 'jjhttpd' | 4 |
| NETGEAR | 'apcfg_mgr' | 96 | Trendnet | 'init' | 3 |
| NETGEAR | 'upnpd' | 95 | Trendnet | 'boa' | 3 |
| NETGEAR | 'rc' | 94 | Trendnet | 'hiawatha' | 2 |
| NETGEAR | 'mysqlslap' | 54 | Trendnet | 'p2p_server' | 2 |
| NETGEAR | 'exim4' | 54 | Trendnet | 'ZNMPClient' | 2 |
| NETGEAR | 'upload.cgi' | 48 | Trendnet | 'pppd_for_pptp' | 2 |
| NETGEAR | 'mailsend' | 43 | Trendnet | 'watchquagga' | 2 |
| NETGEAR | 'pppd_brcm' | 34 | Trendnet | 'tacacs_plus' | 1 |
| NETGEAR | 'afpd' | 32 | Trendnet | 'autoprovision' | 1 |
| NETGEAR | 'dimclient' | 31 | Trendnet | 'router' | 1 |
| NETGEAR | 'wnc_comm' | 29 | Trendnet | 'ctorrent' | 1 |
| NETGEAR | 'vipsecureConfig' | 27 | Trendnet | 'agent' | 1 |
| NETGEAR | 'net-cgi' | 23 | Trendnet | 'packetforge-ng' | 1 |
| NETGEAR | 'gen_password' | 23 | Trendnet | 'pppd-rtk' | 1 |
| NETGEAR | 'ipmitool' | 16 | Trendnet | 'tb_tr069' | 1 |
| NETGEAR | 'watchquagga' | 14 | Trendnet | 'ripngd' | 1 |
| NETGEAR | 'puipv6autodetect' | 14 | Trendnet | 'main.cgi' | 1 |
| NETGEAR | 'rcagentd.svn-base' | 13 | Trendnet | 'bgpd' | 1 |
| NETGEAR | 'spmd' | 12 | Trendnet | 'accountd' | 1 |
| NETGEAR | 'shgw_watchdogd' | 12 | Trendnet | 'httpd' | 1 |
| NETGEAR | 'fcron' | 11 | Ubiquiti | 'udapi-bridge' | 1010 |
| NETGEAR | 'appliance_mgr' | 10 | Ubiquiti | 'cgi' | 470 |
| NETGEAR | 'tinc' | 10 | Ubiquiti | 'basic_radius_auth' | 268 |
| NETGEAR | 'tinctop' | 10 | Ubiquiti | 'udapi-server' | 255 |
| NETGEAR | 'fw-checking' | 8 | Ubiquiti | 'lighttpd' | 243 |
| NETGEAR | 'SkipjamMenus.exe' | 7 | Ubiquiti | 'bgpd' | 231 |
| NETGEAR | '663201' | 4 | Ubiquiti | 'squid3' | 217 |
| NETGEAR | '66E03E' | 4 | Ubiquiti | 'monit' | 108 |
| NETGEAR | '684FE5' | 4 | Ubiquiti | 'switchdrvr' | 72 |
| NETGEAR | '672D95' | 4 | Ubiquiti | 'ripngd' | 37 |
| NETGEAR | 'bst_daemon' | 3 | Ubiquiti | 'lcmd' | 27 |
| NETGEAR | 'funjsq_dl' | 3 | Ubiquiti | 'mcad' | 19 |
| NETGEAR | 'parserd' | 2 | Ubiquiti | 'dirmngr' | 18 |
| NETGEAR | 'fing_dil' | 2 | Ubiquiti | 'postgres' | 16 |
| NETGEAR | 'rclient' | 2 | Ubiquiti | 'rpsd' | 9 |
| NETGEAR | '5B1D1A' | 1 | Ubiquiti | 'miniupnpd' | 7 |
| NETGEAR | '53F662' | 1 | Ubiquiti | 'httpd' | 3 |
| NETGEAR | '7D6E3E' | 1 | Ubiquiti | 'ubnt_displayd' | 3 |
| NETGEAR | '69A460' | 1 | Ubiquiti | 'fwupdate' | 3 |
| NETGEAR | '6561F8' | 1 | Ubiquiti | 'cfgupdate' | 2 |
| NETGEAR | '17AFEC' | 1 | Western-Digital | 'smtp-sink' | 5 |
| NETGEAR | '7812BB' | 1 | Western-Digital | 'monit' | 5 |
| NETGEAR | '66EC8D' | 1 | Western-Digital | 'qmqp-source' | 5 |
| NETGEAR | 'mongoose' | 1 | Western-Digital | 'smtp-source' | 5 |
| NETGEAR | 'swiapp' | 1 | Western-Digital | 'dirmngr' | 5 |

| | | | | | |
|---|---|---|---|---|---|
| NETGEAR | '1B59E4' | 1 | Western-Digital | 'dhclient' | 1 |
| NETGEAR | '1884A9' | 1 | Xiaomi | 'pluginControllor' | 222 |
| NETGEAR | '2292D7' | 1 | Xiaomi | 'datacenter' | 177 |
| NETGEAR | 'aws_json' | 1 | Xiaomi | 'securitypage' | 137 |
| NETGEAR | 'CcspCrSsp' | 1 | Xiaomi | 'apk_query' | 128 |
| NETGEAR | 'check_fw' | 1 | Xiaomi | 'kr_query' | 109 |
| NETGEAR | 'dhclient' | 1 | Xiaomi | 'ustackd' | 104 |
| NETGEAR | 'acos_usbd' | 1 | Xiaomi | 'etm' | 95 |
| NETGEAR | 'cli' | 1 | Xiaomi | 'tquery' | 81 |
| Netis | 'switch' | 4 | Xiaomi | 'cachecenter' | 79 |
| Netis | 'boa' | 2 | Xiaomi | 'StatPoints' | 78 |
| Netis | 'miniupnpd' | 1 | Xiaomi | 'ss-local' | 68 |
| Planet | 'hiawatha' | 47 | Xiaomi | 'elink' | 58 |
| Planet | 'monit' | 24 | Xiaomi | 'wrsst' | 52 |
| Planet | 'thttpd' | 17 | Xiaomi | 'miniupnpd' | 52 |
| Planet | 'ProDaemon' | 15 | Xiaomi | 'samba_multicall' | 51 |
| Planet | 'zebra' | 10 | Xiaomi | 'indexservice' | 45 |
| Planet | 'MAIL.VideoServer.strip' | 6 | Xiaomi | 'mtd_crash_log' | 18 |
| Planet | 'mg_ipinst' | 5 | Xiaomi | 'cdn_conf' | 4 |
| Planet | 'asterisk' | 4 | Xiaomi | 'dnsfixd' | 2 |
| Planet | 'exec_route' | 4 | Xiaomi | 'plugincenter' | 2 |
| Planet | 'gener.cgi' | 4 | Xiaomi | '23770AA' | 1 |
| Planet | 'pptp.cgi' | 4 | Xiaomi | '1F7E27A' | 1 |
| Planet | 'trunk_cmd' | 4 | Xiaomi | '2182FAA' | 1 |
| Planet | 'pptpfw' | 4 | Zyxel | 'capwap_client' | 335 |
| Planet | 'winmsg' | 4 | Zyxel | 'lighttpd' | 294 |
| Planet | 'GBhandler' | 4 | Zyxel | 'zebra' | 239 |
| Planet | 'wan_daemon' | 4 | Zyxel | 'mailsend' | 179 |
| Planet | 'htdbm' | 4 | Zyxel | 'tr69c' | 111 |
| Planet | 'tr69c' | 3 | Zyxel | 'nccconnd' | 94 |
| Planet | 'autop.exe' | 3 | Zyxel | 'ztr69' | 74 |
| Planet | 'ConfigManApp.com' | 3 | Zyxel | 'login.cgi' | 72 |
| Planet | 'logic' | 3 | Zyxel | 'Clicktocontinue.cgi' | 72 |
| Planet | 'ipinstal' | 3 | Zyxel | 'dservice' | 66 |
| Planet | 'eventproc' | 3 | Zyxel | 'social_login.cgi' | 54 |
| Planet | 'test_ap' | 3 | Zyxel | 'auto_add_user' | 54 |
| Planet | 'httpd' | 3 | Zyxel | 'cloudauthd' | 54 |
| Planet | 'onvifServer' | 3 | Zyxel | 'capwap_srv' | 47 |
| Planet | 'httpsrvpwd' | 3 | Zyxel | 'htdbm' | 47 |
| Planet | 'pppd245' | 2 | Zyxel | 'bk_perl' | 46 |
| Planet | 'cwmpd' | 2 | Zyxel | 'racoon' | 45 |
| Planet | 'prog.cgi' | 2 | Zyxel | 'fadd' | 40 |
| Planet | 'pppdo' | 2 | Zyxel | 'tr69' | 39 |
| Planet | 'sendReport' | 2 | Zyxel | 'vpppd' | 39 |
| Planet | 'dccupdate' | 2 | Zyxel | 'radiusc' | 34 |
| Planet | 'HA' | 2 | Zyxel | 'trace' | 32 |
| Planet | 'htmldoc' | 2 | Zyxel | 'httpd' | 32 |
| Planet | 'pppoecd' | 2 | Zyxel | 'zhttpd' | 29 |
| Planet | 'SystemServer' | 2 | Zyxel | 'zyecho_client' | 26 |
| Planet | 'boa' | 2 | Zyxel | 'zyxel_xmpp_client' | 13 |
| Planet | 'hi3518' | 2 | Zyxel | 'zapiBLEService' | 13 |
| Planet | 'panod' | 2 | Zyxel | 'pdbtool' | 9 |
| Planet | 'aistreamer' | 2 | Zyxel | 'sipclient' | 8 |
| Planet | 'badblocks' | 1 | Zyxel | 'APPNotification' | 7 |
| Planet | 'dma' | 1 | Zyxel | 'zytr069main' | 6 |
| Planet | 'freshsnort' | 1 | Zyxel | 'bgpd' | 6 |
| Planet | 'p3scan' | 1 | Zyxel | 'dma' | 6 |
| Planet | 'perl' | 1 | Zyxel | 'sippxy.elf' | 5 |
| QNAP | 'slapd-bind' | 23 | Zyxel | 'wmgeniesrv' | 5 |
| QNAP | 'mount.davfs' | 23 | Zyxel | 'hlasd' | 5 |
| QNAP | 'utilRequest.cgi' | 22 | Zyxel | 'squid' | 5 |
| QNAP | 'badblocks' | 11 | Zyxel | 'tools_mpt.cgi' | 5 |
| QNAP | '638C43' | 1 | Zyxel | 'snmpd' | 4 |
| QNAP | '6C6CF5' | 1 | Zyxel | 'cli' | 3 |
| QNAP | '69F2D0' | 1 | Zyxel | 'pppd_3g' | 3 |
| QNAP | '6B221E' | 1 | Zyxel | 'wsccmd' | 3 |
| Rotek | 'radvd' | 1 | Zyxel | 'zySAS' | 2 |

| | | | | | |
|---|---|---|---|---|---|
| Synology | ’locktest’ | 157 | Zyxel | ’cfm’ | 2 |
| Synology | ’gentest’ | 157 | Zyxel | ’lte_srv_diag’ | 2 |
| Synology | ’masktest’ | 157 | Zyxel | ’smart-polling-service’ | 2 |
| Synology | ’hostapd’ | 141 | Zyxel | ’RMS_monitor’ | 2 |
| Synology | ’nsupdate’ | 135 | Zyxel | ’boa’ | 2 |
| Synology | ’synorelayd’ | 131 | Zyxel | ’ag’ | 2 |
| Synology | ’synosearchagent’ | 130 | Zyxel | ’sysd’ | 2 |
| Synology | ’share-hook’ | 114 | Zyxel | ’auto-ip’ | 1 |
| Synology | ’findhostd’ | 113 | Zyxel | ’ripngd’ | 1 |
| Synology | ’volume-hook’ | 112 | Zyxel | ’uplink_qos’ | 1 |
| Synology | ’usb-copyd’ | 112 | Zyxel | ’wireless.cgi’ | 1 |

Table A.52: Violated Binaries discovered for Pseudo Random Number Generators (*PRNGs*) **R1** and **R2**, entry and possible $\phi$ case.

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| ***K4*** - ‘Weak’ underlying hash function on a KDF/PBE | | | ***K1*** - Constant Passwords on a KDF/PBE | | |
| ASUS | ’busybox’ | 270 | ASUS | ’rc’ | 584 |
| ASUS | ’zebra’ | 17 | ASUS | ’mtd-write’ | 244 |
| ASUS | ’ripd’ | 17 | ASUS | ’qcmap_auth’ | 2 |
| ASUS | ’rc’ | 6 | Arris | ’sc_zipen’ | 4 |
| ASUS | ’login.shadow’ | 2 | D-Link | ’smm’ | 121 |
| Actiontec | ’cm_logic’ | 1 | D-Link | ’commander’ | 9 |
| Alfa | ’busybox’ | 6 | D-Link | ’admin.cgi’ | 6 |
| Arris | ’uhttpd’ | 3 | LinkSys | ’eurl’ | 45 |
| Belkin | ’busybox’ | 4 | LinkSys | ’main_bin’ | 2 |
| Belkin | ’tinylogin’ | 2 | NETGEAR | ’smm’ | 27 |
| Belkin | ’cfm’ | 1 | NETGEAR | ’cli’ | 1 |
| Buffalo | ’busybox’ | 1 | QNAP | ’change_password.cgi’ | 23 |
| D-Link | ’busybox’ | 200 | QNAP | ’authLogin.cgi’ | 6 |
| D-Link | ’sslvpnConfig’ | 122 | QNAP | ’6EC27A’ | 1 |
| D-Link | ’smm’ | 121 | QNAP | ’6D1AB2’ | 1 |
| D-Link | ’admin.cgi’ | 86 | QNAP | ’638C43’ | 1 |
| D-Link | ’tinylogin’ | 38 | QNAP | ’6C6CF5’ | 1 |
| D-Link | ’cm_logic’ | 18 | QNAP | ’69F2D0’ | 1 |
| D-Link | ’login’ | 6 | QNAP | ’6B221E’ | 1 |
| D-Link | ’logic’ | 4 | TP-Link | ’qcmap_auth’ | 34 |
| D-Link | ’uhttpd’ | 4 | TP-Link | ’pure-pw’ | 15 |
| D-Link | ’pure-ftpd’ | 4 | Trendnet | ’daemon_fsp_app’ | 16 |
| D-Link | ’ftpd’ | 4 | Trendnet | ’goahead’ | 2 |
| D-Link | ’cmd’ | 2 | Trendnet | ’cwsysd’ | 1 |
| D-Link | ’rc’ | 2 | Trendnet | ’main.cgi’ | 1 |
| D-Link | ’httpd’ | 1 | Trendnet | ’accountd’ | 1 |
| D-Link | ’tssa’ | 1 | Zyxel | ’zcmd’ | 1 |
| DrayTek | ’busybox’ | 131 | ***K2*** - Constant salt or no salts on a KDF/PBE | | |
| DrayTek | ’mainfunction.cgi’ | 35 | ASUS | ’busybox’ | 270 |
| EdiMax | ’busybox’ | 67 | ASUS | ’rc’ | 6 |
| EdiMax | ’cfg_manager’ | 5 | ASUS | ’login.shadow’ | 2 |
| EdiMax | ’ftpd’ | 5 | Actiontec | ’cm_logic’ | 1 |
| EdiMax | ’boa’ | 4 | Alfa | ’busybox’ | 6 |
| EdiMax | ’startup’ | 4 | Arris | ’uhttpd’ | 3 |
| EdiMax | ’rpcd’ | 2 | Belkin | ’busybox’ | 4 |
| EdiMax | ’telnetd’ | 1 | Belkin | ’tinylogin’ | 2 |
| Inmarsat | ’unix_chkpwd’ | 3 | Belkin | ’cfm’ | 1 |
| Inmarsat | ’unix_update’ | 3 | Buffalo | ’busybox’ | 1 |
| LinkSys | ’eurl’ | 45 | D-Link | ’busybox’ | 200 |
| LinkSys | ’busybox’ | 41 | D-Link | ’sslvpnConfig’ | 122 |
| LinkSys | ’rpcd’ | 3 | D-Link | ’smm’ | 121 |
| LinkSys | ’admin.cgi’ | 2 | D-Link | ’tinylogin’ | 38 |
| LinkSys | ’main_bin’ | 2 | D-Link | ’cm_logic’ | 18 |
| LinkSys | ’uhttpd’ | 1 | D-Link | ’login’ | 6 |
| LinkSys | ’boa’ | 1 | D-Link | ’logic’ | 4 |
| LinkSys | ’rc’ | 1 | D-Link | ’uhttpd’ | 4 |
| LinkSys | ’tinylogin’ | 1 | D-Link | ’ftpd’ | 4 |
| NETGEAR | ’unix_chkpwd’ | 4213 | D-Link | ’cmd’ | 2 |

| | | | | | |
|---|---|---|---|---|---|
| NETGEAR | 'unix_update' | 4213 | D-Link | 'rc' | 2 |
| NETGEAR | 'busybox' | 378 | D-Link | 'httpd' | 1 |
| NETGEAR | 'uhttpd' | 141 | D-Link | 'tssa' | 1 |
| NETGEAR | 'rc' | 61 | DrayTek | 'busybox' | 131 |
| NETGEAR | 'bftpd' | 55 | DrayTek | 'mainfunction.cgi' | 35 |
| NETGEAR | 'appliance_mgr_cli' | 27 | EdiMax | 'busybox' | 67 |
| NETGEAR | 'smm' | 27 | EdiMax | 'cfg_manager' | 5 |
| NETGEAR | 'rpcd' | 26 | EdiMax | 'ftpd' | 5 |
| NETGEAR | 'ngadmin.cgi' | 21 | EdiMax | 'boa' | 4 |
| NETGEAR | 'httpd' | 17 | EdiMax | 'startup' | 4 |
| NETGEAR | 'pam_extrausers_update' | 16 | EdiMax | 'telnetd' | 1 |
| NETGEAR | 'pam_extrausers_chkpwd' | 16 | LinkSys | 'eurl' | 45 |
| NETGEAR | 'screen' | 16 | LinkSys | 'busybox' | 41 |
| NETGEAR | 'login' | 6 | LinkSys | 'main_bin' | 2 |
| NETGEAR | 'cm_logic' | 5 | LinkSys | 'uhttpd' | 1 |
| NETGEAR | 'lc_up' | 2 | LinkSys | 'boa' | 1 |
| NETGEAR | 'mongoose' | 1 | LinkSys | 'rc' | 1 |
| NETGEAR | 'E79B6' | 1 | LinkSys | 'tinylogin' | 1 |
| NETGEAR | 'password_crypt' | 1 | NETGEAR | 'busybox' | 378 |
| Netis | 'busybox' | 96 | NETGEAR | 'uhttpd' | 141 |
| Netis | 'login' | 2 | NETGEAR | 'rc' | 61 |
| Netis | 'startup' | 2 | NETGEAR | 'appliance_mgr_cli' | 27 |
| Netis | 'boa' | 2 | NETGEAR | 'smm' | 27 |
| Planet | 'uhttpd' | 49 | NETGEAR | 'httpd' | 17 |
| Planet | 'busybox' | 10 | NETGEAR | 'login' | 6 |
| Planet | 'cgiMain' | 8 | NETGEAR | 'cm_logic' | 5 |
| Planet | 'login' | 6 | NETGEAR | 'E79B6' | 1 |
| Planet | 'unix_update' | 4 | NETGEAR | 'password_crypt' | 1 |
| Planet | 'unix_chkpwd' | 4 | Netis | 'busybox' | 96 |
| Planet | 'st4YNLn2' | 3 | Netis | 'startup' | 2 |
| Planet | 'swctrl' | 3 | Netis | 'boa' | 2 |
| Planet | 'httpd' | 2 | Planet | 'uhttpd' | 49 |
| Planet | 'startup' | 1 | Planet | 'busybox' | 10 |
| QNAP | '6EC27A' | 1 | Planet | 'cgiMain' | 8 |
| QNAP | '6D1AB2' | 1 | Planet | 'login' | 5 |
| QNAP | '638C43' | 1 | Planet | 'st4YNLn2' | 3 |
| QNAP | '6C6CF5' | 1 | Planet | 'swctrl' | 3 |
| QNAP | '69F2D0' | 1 | Planet | 'httpd' | 2 |
| QNAP | '6B221E' | 1 | Planet | 'startup' | 1 |
| Synology | 'unix_chkpwd' | 176 | QNAP | '6EC27A' | 1 |
| Synology | 'getty' | 128 | QNAP | '6D1AB2' | 1 |
| Synology | 'synouser' | 43 | QNAP | '638C43' | 1 |
| Synology | 'afpd' | 38 | QNAP | '6C6CF5' | 1 |
| Synology | 'findhostd' | 37 | QNAP | '69F2D0' | 1 |
| Synology | 'synorcvol' | 37 | QNAP | '6B221E' | 1 |
| Synology | 'rsrcmonitor.cgi' | 36 | Synology | 'synorcvol' | 37 |
| Synology | 'sftpd' | 17 | Synology | 'sftpd' | 17 |
| Synology | 'manutild' | 7 | TP-Link | 'uhttpd' | 675 |
| Synology | 'synocheckshare' | 7 | TP-Link | 'busybox' | 88 |
| Synology | 'scemd' | 5 | TP-Link | 'login.shadow' | 33 |
| Synology | 'syno-cloud-syncd' | 2 | TP-Link | 'chsh.shadow' | 15 |
| Synology | 'cloud-sync-encrypt-tool' | 2 | TP-Link | 'passwd.shadow' | 15 |
| Synology | 'cloud-sync-starter' | 2 | TP-Link | 'gpasswd' | 15 |
| Synology | 'mysqld' | 1 | TP-Link | 'su' | 15 |
| TP-Link | 'uhttpd' | 675 | TP-Link | 'newusers' | 15 |
| TP-Link | 'busybox' | 88 | TP-Link | 'chgpasswd' | 15 |
| TP-Link | 'pure-pw' | 68 | TP-Link | 'chpasswd.shadow' | 15 |
| TP-Link | 'login.shadow' | 33 | TP-Link | 'chfn.shadow' | 15 |
| TP-Link | 'rpcd' | 3 | TP-Link | 'pure-pw' | 15 |
| TP-Link | 'mysqld' | 1 | Tenda | 'busybox' | 48 |
| TP-Link | 'mariabackup' | 1 | Tenda | 'uhttpd' | 4 |
| Tenda | 'busybox' | 48 | Tenvis | 'busybox' | 2 |
| Tenda | 'uhttpd' | 4 | Thuraya | 'uhttpd' | 2 |
| Tenvis | 'busybox' | 2 | Totolink | 'busybox' | 2 |
| Thuraya | 'uhttpd' | 2 | Totolink | 'uhttpd' | 1 |
| Totolink | 'rpcd' | 4 | Trendnet | 'busybox' | 48 |
| Totolink | 'busybox' | 2 | Trendnet | 'uhttpd' | 26 |

| | | | | | |
|---|---|---|---|---|---|
| Totolink | ’uhttpd’ | 1 | Trendnet | ’daemon_fsp_app’ | 16 |
| Trendnet | ’busybox’ | 48 | Trendnet | ’startup’ | 3 |
| Trendnet | ’uhttpd’ | 26 | Trendnet | ’boa’ | 3 |
| Trendnet | ’daemon_fsp_app’ | 16 | Trendnet | ’tinylogin’ | 2 |
| Trendnet | ’rpcd’ | 7 | Trendnet | ’ftpd’ | 2 |
| Trendnet | ’startup’ | 3 | Trendnet | ’goahead’ | 2 |
| Trendnet | ’boa’ | 3 | Trendnet | ’init’ | 1 |
| Trendnet | ’tinylogin’ | 2 | Trendnet | ’logic’ | 1 |
| Trendnet | ’ftpd’ | 2 | Trendnet | ’cwsysd’ | 1 |
| Trendnet | ’goahead’ | 2 | Ubiquiti | ’uhttpd’ | 78 |
| Trendnet | ’init’ | 1 | Xiaomi | ’uhttpd’ | 299 |
| Trendnet | ’logic’ | 1 | Zyxel | ’busybox’ | 106 |
| Trendnet | ’cwsysd’ | 1 | Zyxel | ’makepwd’ | 66 |
| Ubiquiti | ’unix_chkpwd’ | 354 | Zyxel | ’uhttpd’ | 36 |
| Ubiquiti | ’unix_update’ | 343 | Zyxel | ’mini_httpd’ | 13 |
| Ubiquiti | ’uhttpd’ | 78 | Zyxel | ’cfg_manager’ | 3 |
| Ubiquiti | ’basic_nis_auth’ | 69 | Zyxel | ’boa’ | 2 |
| Ubiquiti | ’sulogin’ | 24 | Zyxel | ’startup’ | 2 |
| Ubiquiti | ’rpcd’ | 3 | Zyxel | ’wireless.cgi’ | 1 |
| Western-Digital | ’unix_chkpwd’ | 5 | ***K3*** - ‘Weak’ number of iteration on a KDF/PBE | | |
| Western-Digital | ’unix_update’ | 5 | Arris | ’sc_zipen’ | 4 |
| Western-Digital | ’sulogin’ | 4 | DrayTek | ’fw_printenv’ | 62 |
| Western-Digital | ’screen’ | 3 | LinkSys | ’eurl’ | 45 |
| Western-Digital | ’pure-ftpd’ | 1 | NETGEAR | ’lc_up’ | 2 |
| Xiaomi | ’uhttpd’ | 299 | NETGEAR | ’mongoose’ | 1 |
| Xiaomi | ’su’ | 54 | Synology | ’syno-cloud-syncd’ | 2 |
| Zyxel | ’busybox’ | 131 | Synology | ’cloud-sync-encrypt-tool’ | 2 |
| Zyxel | ’unix_update’ | 80 | Synology | ’cloud-sync-starter’ | 2 |
| Zyxel | ’unix_chkpwd’ | 80 | Synology | ’mysqld’ | 1 |
| Zyxel | ’makepwd’ | 66 | TP-Link | ’mysqld’ | 1 |
| Zyxel | ’uhttpd’ | 36 | TP-Link | ’mariabackup’ | 1 |
| Zyxel | ’pure-ftpd’ | 32 | Tenda | ’ucloud’ | 4 |
| Zyxel | ’mini_httpd’ | 13 | Trendnet | ’daemon_fsp_app’ | 16 |
| Zyxel | ’ripd’ | 7 | Ubiquiti | ’ubntbox’ | 76 |
| Zyxel | ’zebra’ | 7 | Xiaomi | ’ss-local’ | 68 |
| Zyxel | ’cfg_manager’ | 3 | Xiaomi | ’ss-redir’ | 54 |
| Zyxel | ’boa’ | 2 | Xiaomi | ’ss-tunnel’ | 3 |
| Zyxel | ’startup’ | 2 | Zyxel | ’zycfgfilter’ | 90 |
| Zyxel | ’wireless.cgi’ | 1 | Zyxel | ’zcmd’ | 74 |

Table A.53: Violated Binaries discovered for Key Derivation Functions (*KDFs*) and Password Based Encryption (*PBE*) rules **K1**, **K2**, **K3** and **K4**, entry and possible $\phi$ case.

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| ***M2*** - ‘Weak’ underlying hash function on a MAC | | | ***M1*** - Constant Encryption/Decryption Keys on a MAC | | |
| ASUS | ’wpa_supplicant’ | 152 | DrayTek | ’tr069_client’ | 150 |
| ASUS | ’hostapd’ | 95 | NETGEAR | ’Netgear_ddns’ | 144 |
| ASUS | ’wpa_supplicant-2.7’ | 29 | NETGEAR | ’httpd’ | 24 |
| ASUS | ’racoon’ | 2 | NETGEAR | ’ntgrddns’ | 14 |
| AVM | ’hostapd’ | 5 | ***M3*** - Non-secure key length on a MAC function | | |
| AVM | ’wpa_supplicant’ | 5 | ASUS | ’hostapd’ | 29 |
| Actiontec | ’racoon’ | 2 | Alfa | ’wpad’ | 8 |
| Alfa | ’wpad’ | 33 | LinkSys | ’hostapd’ | 1 |
| D-Link | ’prog-cgi’ | 126 | NETGEAR | ’dimclient’ | 31 |
| D-Link | ’mdb’ | 60 | NETGEAR | ’hostapd’ | 1 |
| D-Link | ’wpa_supplicant’ | 23 | Synology | ’hostapd’ | 1 |
| D-Link | ’snmpd’ | 9 | TP-Link | ’hostapd’ | 7 |
| D-Link | ’snmptrap’ | 7 | Tenda | ’hostapd’ | 1 |
| D-Link | ’hostapd’ | 7 | Totolink | ’tincd’ | 1 |
| D-Link | ’dam’ | 4 | - | - | - |
| D-Link | ’racoon’ | 4 | - | - | - |
| D-Link | ’xsupplicant’ | 4 | - | - | - |
| D-Link | ’prog.cgi’ | 3 | - | - | - |
| Dell | ’ciphertool’ | 41 | - | - | - |
| DrayTek | ’onvif_func’ | 18 | - | - | - |

| | | | | | |
|---|---|---|---|---|---|
| DrayTek | 'hostapd' | 5 | - | - | - |
| DrayTek | 'wpa_supplicant' | 5 | - | - | - |
| EdiMax | 'device_service' | 7 | - | - | - |
| EdiMax | 'racoon' | 3 | - | - | - |
| EdiMax | 'wpa_supplicant' | 2 | - | - | - |
| EdiMax | 'hostapd' | 2 | - | - | - |
| Inmarsat | 'hostapd' | 3 | - | - | - |
| LinkSys | 'dhclient' | 71 | - | - | - |
| LinkSys | 'wpa_supplicant' | 11 | - | - | - |
| LinkSys | 'hostapd' | 9 | - | - | - |
| LinkSys | 'onvif1.0' | 4 | - | - | - |
| LinkSys | 'onvif2' | 4 | - | - | - |
| MicroTik | 'ipsec' | 350 | - | - | - |
| NETGEAR | 'wpa_supplicant' | 69 | - | - | - |
| NETGEAR | 'hostapd' | 43 | - | - | - |
| NETGEAR | 'wpa_supplicant-macsec' | 1 | - | - | - |
| Planet | 'snmpd' | 50 | - | - | - |
| Planet | 'wpa_supplicant' | 27 | - | - | - |
| Planet | 'prog.cgi' | 2 | - | - | - |
| Planet | 'boa' | 2 | - | - | - |
| Planet | 'xsupplicant' | 1 | - | - | - |
| Planet | 'snmptrap' | 1 | - | - | - |
| QNAP | 'wpa_supplicant' | 13 | - | - | - |
| Synology | 'wpa_supplicant' | 160 | - | - | - |
| Synology | 'hostapd' | 145 | - | - | - |
| Synology | 'git-imap-send' | 4 | - | - | - |
| TP-Link | 'wpa_supplicant' | 80 | - | - | - |
| TP-Link | 'racoon' | 56 | - | - | - |
| TP-Link | 'chm' | 49 | - | - | - |
| TP-Link | 'hostapd' | 28 | - | - | - |
| Tenda | 'racoon' | 22 | - | - | - |
| Tenda | 'hostapd' | 2 | - | - | - |
| Tenda | 'wpa_supplicant' | 1 | - | - | - |
| Tenvis | 'onvif' | 1 | - | - | - |
| Totolink | 'wpa_supplicant' | 4 | - | - | - |
| Trendnet | 'wpa_supplicant' | 18 | - | - | - |
| Trendnet | 'hostapd' | 2 | - | - | - |
| Trendnet | 'airdecap-ng' | 1 | - | - | - |
| Trendnet | 'aircrack-ng' | 1 | - | - | - |
| Ubiquiti | 'wpad' | 264 | - | - | - |
| Ubiquiti | 'hostapd' | 39 | - | - | - |
| Ubiquiti | 'wpa_supplicant' | 37 | - | - | - |
| Ubiquiti | 'snmpd' | 20 | - | - | - |
| Western-Digital | 'wpa_supplicant' | 4 | - | - | - |
| Western-Digital | 'hostapd' | 3 | - | - | - |
| Xiaomi | 'wpa_supplicant' | 13 | - | - | - |
| Xiaomi | '245506E' | 1 | - | - | - |
| Zyxel | 'racoon' | 21 | - | - | - |
| Zyxel | 'radclient' | 15 | - | - | - |
| Zyxel | 'radeapclient' | 15 | - | - | - |
| Zyxel | 'radiusd' | 6 | - | - | - |
| Zyxel | 'hostapd' | 1 | - | - | - |
| Zyxel | 'wpa_supplicant-macsec' | 1 | - | - | - |
| Zyxel | 'wpa_supplicant' | 1 | - | - | - |

Table A.54: Violated Binaries discovered for Message Authentication Codes (*MACs*) rules **M1**, **M2** and **M3**, entry and possible $\phi$ case.

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| ***P1*** - Usage of insecure RSA encryption padding schemes | | | ***P3*** - X.509 certificate usage of 'weak' digest function | | |
| ASUS | 'cfg_client' | 335 | ASUS | 'qmi_ip' | 1 |
| ASUS | 'cfg_server' | 14 | D-Link | 'mpop' | 85 |
| Alfa | 'tor' | 2 | D-Link | 'x509SelfSign' | 75 |
| Alfa | 'rsa_test' | 1 | D-Link | 'mapd' | 28 |
| D-Link | 'captival_portal' | 19 | D-Link | 'gencert' | 5 |

| | | | | | |
|---|---|---|---|---|---|
| D-Link | 'httpd' | 5 | D-Link | 'imspector' | 2 |
| D-Link | 'shareport' | 4 | LinkSys | 'mapd' | 2 |
| D-Link | 'spt' | 4 | NETGEAR | 'ntfsdecrypt' | 860 |
| D-Link | 'EmbedThunderManager' | 1 | NETGEAR | 'certgen' | 222 |
| LinkSys | 'iperf3' | 1 | NETGEAR | 'x509SelfSign' | 27 |
| NETGEAR | 'fvdropbox' | 2244 | Planet | 'monit' | 24 |
| NETGEAR | 'avdu' | 2014 | Synology | 'lftp' | 88 |
| NETGEAR | 'fvamazon' | 1232 | Synology | 'nzbget' | 18 |
| NETGEAR | 'readynasd' | 928 | Synology | 'ncat' | 2 |
| NETGEAR | 'NetReadyAgent' | 12 | TP-Link | 'httpd' | 47 |
| NETGEAR | 'tincd' | 10 | TP-Link | 'sslselfsign' | 15 |
| NETGEAR | 'TPMFactoryUpd' | 8 | TP-Link | 'mysqlimport' | 1 |
| NETGEAR | 'dimclient' | 4 | TP-Link | 'mysqlcheck' | 1 |
| NETGEAR | 'iperf3' | 1 | TP-Link | 'mysqlshow' | 1 |
| Synology | 'synolicense_uninstall' | 52 | TP-Link | 'mysqldump' | 1 |
| Synology | 'sftpd' | 17 | TP-Link | 'mysqlslap' | 1 |
| Synology | 'synoddsm-hostd' | 3 | TP-Link | 'mysqladmin' | 1 |
| TP-Link | 'httpd' | 36 | TP-Link | 'mysql' | 1 |
| TP-Link | 'eap-mesh' | 12 | TP-Link | 'mysqlbinlog' | 1 |
| TP-Link | 'eapcs' | 8 | TP-Link | 'mysqltest' | 1 |
| TP-Link | 'o_p_test' | 4 | Trendnet | 'ipheth-pair' | 1 |
| TP-Link | 'rsa_decrypt' | 4 | Ubiquiti | 'monit' | 108 |
| TP-Link | 'tdpServer' | 3 | Ubiquiti | 'httping' | 3 |
| Tenda | 'eventdispatcher' | 2 | Western-Digital | 'ncat' | 3 |
| Tenda | 'racoon' | 1 | Western-Digital | 'ntfsdecrypt' | 1 |
| Totolink | 'tincd' | 1 | Xiaomi | 'syslog-ng' | 14 |
| Xiaomi | 'etm' | 95 | - | - | - |
| Xiaomi | 'rsa_test' | 1 | - | - | - |
| Zyxel | 'zhttpd' | 30 | - | - | - |
| Zyxel | 'zyxel_xmpp_client' | 12 | - | - | - |
| Zyxel | 'zyxel_encrypt_hash' | 9 | - | - | - |
| Zyxel | 'httpd' | 5 | - | - | - |

Table A.55: Violated Binaries discovered for Public Key Cryptography rules **P1** and **P3**, entry and not discovered $\phi$ case.

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| | | ***R1*** - PRNG static seed | | | |
| D-Link | 'htpasswd' | 36 | TP-Link | 'sessmngr' | 6 |
| D-Link | 'EmbedThunderManager' | 1 | TP-Link | 'capwap' | 3 |
| LinkSys | 'udhcpd' | 4 | Xiaomi | 'etm' | 95 |
| LinkSys | 'Mercury.snos' | 2 | Zyxel | 'login.cgi' | 84 |
| NETGEAR | 'htpasswd' | 2261 | Zyxel | 'dispatcher.cgi' | 36 |
| NETGEAR | 'tincd' | 10 | Zyxel | 'htpasswd' | 25 |
| NETGEAR | 'iss.exe' | 1 | Zyxel | '433DEB' | 18 |
| Planet | 'htpasswd' | 4 | Zyxel | '443F0F' | 9 |
| Synology | 'htpasswd' | 50 | Zyxel | '4729B2' | 9 |
| TP-Link | 'cet' | 45 | Zyxel | '432C64' | 9 |
| TP-Link | 'iked' | 10 | Zyxel | '472073' | 3 |
| TP-Link | 'aaa' | 6 | - | - | - |
| | | ***R2*** - Low entropy sources for seeds | | | |
| ASUS | 'aaews' | 616 | Synology | 'usb-copyd' | 112 |
| ASUS | 'mastiff' | 510 | Synology | 'usbcopy-hook' | 112 |
| ASUS | 'watchquagga' | 436 | Synology | 'usb-copy-notifier' | 112 |
| ASUS | 'cfg_server' | 367 | Synology | 'usb-copy-starter' | 110 |
| ASUS | 'httpd' | 195 | Synology | 'zip' | 45 |
| ASUS | 'zebra' | 190 | Synology | 'postgres' | 36 |
| ASUS | 'miniupnpd' | 154 | Synology | 'network.cgi' | 36 |
| ASUS | 'rc' | 51 | Synology | 'upgrade.cgi' | 35 |
| ASUS | 'boa' | 16 | Synology | 'thumbnail.cgi' | 35 |
| ASUS | 'cfg_client' | 15 | Synology | 'postgres32' | 34 |
| ASUS | 'btgatt-server' | 8 | Synology | 'fileindexd' | 31 |
| ASUS | 'bluealsa' | 7 | Synology | 'imap-login' | 28 |
| ASUS | 'sip_proxy' | 3 | Synology | 'dovecot-auth' | 28 |
| ASUS | 'tr69c' | 2 | Synology | 'ssl-build-param' | 28 |
| ASUS | 'newusers' | 2 | Synology | 'dovecot' | 28 |

| | | | | | |
|---|---|---|---|---|---|
| AVM | ’mount.davfs’ | 11 | Synology | ’pop3-login’ | 28 |
| AVM | ’cloudmsgd’ | 10 | Synology | ’postlock’ | 27 |
| AVM | ’wlmngr2’ | 2 | Synology | ’pop3’ | 27 |
| AVM | ’tr69c’ | 1 | Synology | ’imap’ | 27 |
| Actiontec | ’zebra’ | 3 | Synology | ’synodisk’ | 24 |
| Actiontec | ’tr69c’ | 2 | Synology | ’scemd’ | 15 |
| Actiontec | ’uhttpd’ | 1 | Synology | ’synobox’ | 14 |
| Actiontec | ’sntp’ | 1 | Synology | ’image_thumb.cgi’ | 8 |
| Actiontec | ’detectWANService’ | 1 | Synology | ’heartbeatd’ | 6 |
| Actiontec | ’cm_logic’ | 1 | Synology | ’synowolagentd’ | 5 |
| Arris | ’ripngd’ | 4 | Synology | ’git-fast-import’ | 4 |
| D-Link | ’pppdo’ | 183 | Synology | ’git-http-push’ | 4 |
| D-Link | ’upnpc-ddns’ | 173 | Synology | ’git-credential-store’ | 4 |
| D-Link | ’vipsecureConfig’ | 143 | Synology | ’git-remote-https’ | 4 |
| D-Link | ’prog-cgi’ | 126 | Synology | ’git-http-fetch’ | 4 |
| D-Link | ’admin.cgi’ | 67 | Synology | ’synodbudd’ | 4 |
| D-Link | ’mailsend’ | 63 | Synology | ’dhclient’ | 4 |
| D-Link | ’da_adaptor’ | 61 | Synology | ’aa_cmd’ | 3 |
| D-Link | ’newp2p’ | 59 | Synology | ’iscsiadm’ | 3 |
| D-Link | ’snmpd’ | 49 | Synology | ’PkgSynoMan.cgi’ | 3 |
| D-Link | ’httpd’ | 46 | Synology | ’synodatacollectd’ | 2 |
| D-Link | ’cgibin’ | 45 | Synology | ’cloud-cleand’ | 2 |
| D-Link | ’mapd’ | 45 | Synology | ’cloud-control’ | 2 |
| D-Link | ’p2p_server’ | 43 | Synology | ’img_backup’ | 2 |
| D-Link | ’shgw_watchdogd’ | 41 | Synology | ’syno-cloud-syncd’ | 2 |
| D-Link | ’zebra’ | 32 | Synology | ’debug’ | 2 |
| D-Link | ’afpd’ | 26 | Synology | ’synologyfilemanager-authd’ | 2 |
| D-Link | ’lighttpd’ | 20 | Synology | ’pgbouncer’ | 2 |
| D-Link | ’perl’ | 20 | Synology | ’dig’ | 2 |
| D-Link | ’jjhttpd’ | 18 | Synology | ’CSTNVolChange’ | 2 |
| D-Link | ’crtmpserver’ | 18 | Synology | ’RestoreNode’ | 2 |
| D-Link | ’linkd.out’ | 14 | Synology | ’cloud-sync-encrypt-tool’ | 2 |
| D-Link | ’test_ap’ | 14 | Synology | ’cloud-cached’ | 2 |
| D-Link | ’dv8_agent’ | 13 | Synology | ’syno-cloud-clientd’ | 2 |
| D-Link | ’op_server’ | 13 | Synology | ’feasibility-check’ | 2 |
| D-Link | ’mt-daapd’ | 11 | Synology | ’cloud-sync-starter’ | 2 |
| D-Link | ’hd_verify’ | 11 | Synology | ’db-check’ | 2 |
| D-Link | ’ipca’ | 10 | Synology | ’cloud-authd’ | 2 |
| D-Link | ’x509SelfSign’ | 9 | Synology | ’syno-letsencrypt’ | 2 |
| D-Link | ’tr69c’ | 9 | Synology | ’synoupgrade’ | 1 |
| D-Link | ’miniupnpd’ | 8 | Synology | ’dms’ | 1 |
| D-Link | ’commander’ | 8 | Synology | ’main.cgi’ | 1 |
| D-Link | ’onvifServer’ | 6 | Synology | ’synotifyd’ | 1 |
| D-Link | ’GBhandler’ | 5 | Synology | ’ha.cgi’ | 1 |
| D-Link | ’prog.cgi’ | 4 | Synology | ’virtual’ | 1 |
| D-Link | ’ripngd’ | 4 | Synology | ’local’ | 1 |
| D-Link | ’tr69’ | 4 | Synology | ’bounce’ | 1 |
| D-Link | ’sudo’ | 4 | TP-Link | ’cloud-brd’ | 628 |
| D-Link | ’webs’ | 4 | TP-Link | ’cloud-client’ | 361 |
| D-Link | ’record_server’ | 3 | TP-Link | ’cet’ | 229 |
| D-Link | ’MAIL.VideoServer.strip’ | 3 | TP-Link | ’uac’ | 200 |
| D-Link | ’watchquagga’ | 2 | TP-Link | ’relayd’ | 174 |
| D-Link | ’ppp’ | 2 | TP-Link | ’miniupnpd’ | 144 |
| D-Link | ’dnsproxy’ | 2 | TP-Link | ’cwmp’ | 109 |
| D-Link | ’vvctl’ | 2 | TP-Link | ’newusers’ | 80 |
| D-Link | ’newgrp’ | 2 | TP-Link | ’pure-pw’ | 67 |
| D-Link | ’resident’ | 2 | TP-Link | ’streamd’ | 58 |
| D-Link | ’hapClient’ | 2 | TP-Link | ’ipcamera’ | 58 |
| D-Link | ’agent’ | 1 | TP-Link | ’uhttpd’ | 40 |
| D-Link | ’accessctl’ | 1 | TP-Link | ’dsd’ | 34 |
| D-Link | ’lprm’ | 1 | TP-Link | ’nvid’ | 32 |
| D-Link | ’lpr’ | 1 | TP-Link | ’nvrcore’ | 20 |
| D-Link | ’lpq’ | 1 | TP-Link | ’onboarding’ | 19 |
| D-Link | ’lpd’ | 1 | TP-Link | ’speaker’ | 19 |
| D-Link | ’lpc’ | 1 | TP-Link | ’storage’ | 14 |
| D-Link | ’EmbedThunderManager’ | 1 | TP-Link | ’cloud-sdk’ | 12 |

| | | | | | |
|---|---|---|---|---|---|
| D-Link | 'tr069' | 1 | TP-Link | 'cloud_brd' | 10 |
| D-Link | 'tssa' | 1 | TP-Link | 'dcd' | 8 |
| Dell | 'compmanager' | 20 | TP-Link | 'eapcs' | 8 |
| DrayTek | 'mainfunction.cgi' | 63 | TP-Link | 'httpd' | 7 |
| DrayTek | 'onvif_func' | 55 | TP-Link | 'predictd' | 7 |
| DrayTek | 'lighttpd' | 46 | TP-Link | 'v6plus' | 6 |
| DrayTek | 'oneTimeCall' | 30 | TP-Link | 'eap-cs' | 5 |
| DrayTek | 'dray_apm' | 16 | TP-Link | 'vod' | 5 |
| DrayTek | 'dhcrelay' | 14 | TP-Link | 'voip_client' | 4 |
| DrayTek | 'acs' | 9 | TP-Link | 'zavim' | 4 |
| DrayTek | 'goahead' | 2 | TP-Link | 'cloud_client' | 3 |
| EdiMax | 'zebra' | 14 | TP-Link | 'tr69c' | 3 |
| EdiMax | 'boa' | 4 | TP-Link | 'zebra' | 3 |
| EdiMax | 'btget' | 3 | TP-Link | 'dig' | 3 |
| EdiMax | 'tr69c' | 3 | TP-Link | 'host' | 3 |
| EdiMax | 'lighttpd' | 2 | TP-Link | 'aria2c' | 3 |
| EdiMax | 'mailsend' | 2 | TP-Link | 'wlan-manager' | 3 |
| FOSCAM | 'jco_server' | 1 | TP-Link | 'mobile' | 2 |
| HP | 'lighttpd' | 5 | TP-Link | 'appcmd' | 2 |
| Inmarsat | 'ogg123' | 3 | TP-Link | 'samba_multicall' | 1 |
| Inmarsat | 'asterisk' | 1 | TP-Link | 'mysqlslap' | 1 |
| Inmarsat | 'lighttpd' | 1 | TP-Link | 'mediaServer' | 1 |
| LinkSys | 'dhclient' | 86 | Tenda | 'pppdForPptp' | 33 |
| LinkSys | 'httpd' | 15 | Tenda | 'xl2tpdpppd' | 33 |
| LinkSys | 'ripngd' | 14 | Tenda | 'tr69c' | 29 |
| LinkSys | 'zebra' | 8 | Tenda | 'httpd' | 16 |
| LinkSys | 'tr69c' | 6 | Tenda | 'portal' | 10 |
| LinkSys | 'fwupd' | 6 | Tenda | 'zebra' | 6 |
| LinkSys | 'dnsproxy' | 4 | Tenda | 'pppdForPppServer' | 5 |
| LinkSys | 'cwmpCPE' | 3 | Tenda | 'pppoa' | 3 |
| LinkSys | 'bgpd' | 3 | Tenda | 'ripngd' | 2 |
| LinkSys | 'lrhkprvsn' | 3 | Tenda | 'bi' | 2 |
| LinkSys | 'admin.cgi' | 2 | Tenda | 'pppd244' | 2 |
| LinkSys | 'ospfd' | 2 | Tenda | 'pppd_3g' | 1 |
| LinkSys | 'ospf6d' | 2 | Tenda | 'pppd_245' | 1 |
| LinkSys | 'mailsend' | 2 | Tenda | 'wlmngr2' | 1 |
| LinkSys | 'boa' | 1 | Tenda | 'pppds' | 1 |
| LinkSys | 'setup.cgi' | 1 | Tenvis | 'tutk' | 2 |
| LinkSys | 'watchquagga' | 1 | Totolink | 'ss-orig-redir' | 6 |
| LinkSys | 'lighttpd' | 1 | Totolink | 'ss-orig-tunnel' | 6 |
| LinkSys | 'LiveviewControlServer' | 1 | Totolink | 'ss-orig-local' | 6 |
| MicroTik | 'ddns' | 580 | Totolink | 'ssr-redir' | 6 |
| MicroTik | 'ipsec' | 513 | Totolink | 'ssr-local' | 6 |
| NETGEAR | 'readyNASVault' | 1790 | Totolink | 'tinc' | 5 |
| NETGEAR | 'mysqlmanager' | 714 | Totolink | 'ppp2d' | 2 |
| NETGEAR | 'htdbm' | 714 | Totolink | 'rc' | 1 |
| NETGEAR | 'httpd' | 581 | Totolink | 'miniupnpd' | 1 |
| NETGEAR | 'zebra' | 382 | Trendnet | 'zebra' | 24 |
| NETGEAR | 'auditd' | 289 | Trendnet | 'mailsend' | 10 |
| NETGEAR | 'rcagentd' | 284 | Trendnet | 'snmpd' | 10 |
| NETGEAR | 'rc_apps' | 185 | Trendnet | 'lighttpd' | 10 |
| NETGEAR | 'Netgear_ddns' | 144 | Trendnet | 'vcm_serv' | 10 |
| NETGEAR | 'ripngd' | 135 | Trendnet | 'hicore' | 6 |
| NETGEAR | 'uhttpd' | 131 | Trendnet | 'webproc' | 4 |
| NETGEAR | 'mini_httpd' | 123 | Trendnet | 'pppds' | 4 |
| NETGEAR | 'miniupnpd' | 119 | Trendnet | 'jjhttpd' | 4 |
| NETGEAR | 'lighttpd' | 116 | Trendnet | 'init' | 3 |
| NETGEAR | 'nlogin.cgi' | 96 | Trendnet | 'boa' | 3 |
| NETGEAR | 'apcomm' | 96 | Trendnet | 'hiawatha' | 2 |
| NETGEAR | 'apcfg_mgr' | 96 | Trendnet | 'p2p_server' | 2 |
| NETGEAR | 'upnpd' | 95 | Trendnet | 'ZNMPClient' | 2 |
| NETGEAR | 'rc' | 94 | Trendnet | 'pppd_for_pptp' | 2 |
| NETGEAR | 'mysqlslap' | 54 | Trendnet | 'watchquagga' | 2 |
| NETGEAR | 'exim4' | 54 | Trendnet | 'tacacs_plus' | 1 |
| NETGEAR | 'upload.cgi' | 48 | Trendnet | 'autoprovision' | 1 |
| NETGEAR | 'mailsend' | 43 | Trendnet | 'router' | 1 |
| NETGEAR | 'pppd_brcm' | 34 | Trendnet | 'ctorrent' | 1 |

| | | | | | |
|---|---|---|---|---|---|
| NETGEAR | 'afpd' | 32 | Trendnet | 'agent' | 1 |
| NETGEAR | 'dimclient' | 31 | Trendnet | 'packetforge-ng' | 1 |
| NETGEAR | 'wnc_comm' | 29 | Trendnet | 'pppd-rtk' | 1 |
| NETGEAR | 'vipsecureConfig' | 27 | Trendnet | 'tb_tr069' | 1 |
| NETGEAR | 'net-cgi' | 23 | Trendnet | 'ripngd' | 1 |
| NETGEAR | 'ipmitool' | 16 | Trendnet | 'main.cgi' | 1 |
| NETGEAR | 'watchquagga' | 14 | Trendnet | 'bgpd' | 1 |
| NETGEAR | 'puipv6autodetect' | 14 | Trendnet | 'accountd' | 1 |
| NETGEAR | 'rcagentd.svn-base' | 13 | Trendnet | 'httpd' | 1 |
| NETGEAR | 'spmd' | 12 | Ubiquiti | 'udapi-bridge' | 1010 |
| NETGEAR | 'shgw_watchdogd' | 12 | Ubiquiti | 'cgi' | 470 |
| NETGEAR | 'fcron' | 11 | Ubiquiti | 'basic_radius_auth' | 268 |
| NETGEAR | 'appliance_mgr' | 10 | Ubiquiti | 'udapi-server' | 255 |
| NETGEAR | 'tinc' | 10 | Ubiquiti | 'lighttpd' | 243 |
| NETGEAR | 'tinctop' | 10 | Ubiquiti | 'bgpd' | 231 |
| NETGEAR | 'fw-checking' | 8 | Ubiquiti | 'squid3' | 217 |
| NETGEAR | 'SkipjamMenus.exe' | 7 | Ubiquiti | 'monit' | 108 |
| NETGEAR | '663201' | 4 | Ubiquiti | 'switchdrvr' | 72 |
| NETGEAR | '66E03E' | 4 | Ubiquiti | 'ripngd' | 37 |
| NETGEAR | '684FE5' | 4 | Ubiquiti | 'lcmd' | 27 |
| NETGEAR | '672D95' | 4 | Ubiquiti | 'mcad' | 19 |
| NETGEAR | 'bst_daemon' | 3 | Ubiquiti | 'dirmngr' | 18 |
| NETGEAR | 'funjsq_dl' | 3 | Ubiquiti | 'postgres' | 16 |
| NETGEAR | 'parserd' | 2 | Ubiquiti | 'rpsd' | 9 |
| NETGEAR | 'fing_dil' | 2 | Ubiquiti | 'miniupnpd' | 7 |
| NETGEAR | 'rclient' | 2 | Ubiquiti | 'httpd' | 3 |
| NETGEAR | '5B1D1A' | 1 | Ubiquiti | 'ubnt_displayd' | 3 |
| NETGEAR | '53F662' | 1 | Ubiquiti | 'fwupdate' | 3 |
| NETGEAR | '7D6E3E' | 1 | Ubiquiti | 'cfgupdate' | 2 |
| NETGEAR | '69A460' | 1 | Western-Digital | 'smtp-sink' | 5 |
| NETGEAR | '6561F8' | 1 | Western-Digital | 'monit' | 5 |
| NETGEAR | '17AFEC' | 1 | Western-Digital | 'qmqp-source' | 5 |
| NETGEAR | '7812BB' | 1 | Western-Digital | 'smtp-source' | 5 |
| NETGEAR | '66EC8D' | 1 | Western-Digital | 'dirmngr' | 5 |
| NETGEAR | 'mongoose' | 1 | Western-Digital | 'dhclient' | 1 |
| NETGEAR | 'swiapp' | 1 | Xiaomi | 'pluginControllor' | 222 |
| NETGEAR | '1B59E4' | 1 | Xiaomi | 'datacenter' | 177 |
| NETGEAR | '1884A9' | 1 | Xiaomi | 'securitypage' | 137 |
| NETGEAR | '2292D7' | 1 | Xiaomi | 'apk_query' | 128 |
| NETGEAR | 'aws_json' | 1 | Xiaomi | 'kr_query' | 109 |
| NETGEAR | 'CcspCrSsp' | 1 | Xiaomi | 'ustackd' | 104 |
| NETGEAR | 'check_fw' | 1 | Xiaomi | 'etm' | 95 |
| NETGEAR | 'dhclient' | 1 | Xiaomi | 'tquery' | 81 |
| NETGEAR | 'acos_usbd' | 1 | Xiaomi | 'cachecenter' | 79 |
| NETGEAR | 'cli' | 1 | Xiaomi | 'StatPoints' | 78 |
| Netis | 'switch' | 4 | Xiaomi | 'ss-local' | 68 |
| Netis | 'boa' | 2 | Xiaomi | 'elink' | 58 |
| Netis | 'miniupnpd' | 1 | Xiaomi | 'wrsst' | 52 |
| Planet | 'hiawatha' | 47 | Xiaomi | 'miniupnpd' | 52 |
| Planet | 'monit' | 24 | Xiaomi | 'samba_multicall' | 51 |
| Planet | 'ProDaemon' | 15 | Xiaomi | 'indexservice' | 45 |
| Planet | 'zebra' | 10 | Xiaomi | 'mtd_crash_log' | 18 |
| Planet | 'MAIL.VideoServer.strip' | 6 | Xiaomi | 'cdn_conf' | 4 |
| Planet | 'mg_ipinst' | 5 | Xiaomi | 'dnsfixd' | 2 |
| Planet | 'asterisk' | 4 | Xiaomi | 'plugincenter' | 2 |
| Planet | 'exec_route' | 4 | Xiaomi | '23770AA' | 1 |
| Planet | 'gener.cgi' | 4 | Xiaomi | '1F7E27A' | 1 |
| Planet | 'pptp.cgi' | 4 | Xiaomi | '2182FAA' | 1 |
| Planet | 'trunk_cmd' | 4 | Zyxel | 'capwap_client' | 335 |
| Planet | 'pptpfw' | 4 | Zyxel | 'lighttpd' | 294 |
| Planet | 'winmsg' | 4 | Zyxel | 'zebra' | 239 |
| Planet | 'GBhandler' | 4 | Zyxel | 'mailsend' | 179 |
| Planet | 'thttpd' | 4 | Zyxel | 'tr69c' | 111 |
| Planet | 'wan_daemon' | 4 | Zyxel | 'nccconnd' | 94 |
| Planet | 'htdbm' | 4 | Zyxel | 'ztr69' | 74 |
| Planet | 'tr69c' | 3 | Zyxel | 'login.cgi' | 72 |
| Planet | 'autop.exe' | 3 | Zyxel | 'Clicktocontinue.cgi' | 72 |

| | | | | | |
|---|---|---|---|---|---|
| Planet | 'ConfigManApp.com' | 3 | Zyxel | 'dservice' | 66 |
| Planet | 'logic' | 3 | Zyxel | 'social_login.cgi' | 54 |
| Planet | 'ipinstal' | 3 | Zyxel | 'auto_add_user' | 54 |
| Planet | 'eventproc' | 3 | Zyxel | 'cloudauthd' | 54 |
| Planet | 'test_ap' | 3 | Zyxel | 'capwap_srv' | 47 |
| Planet | 'httpd' | 3 | Zyxel | 'htdbm' | 47 |
| Planet | 'onvifServer' | 3 | Zyxel | 'bk_perl' | 46 |
| Planet | 'httpsrvpwd' | 3 | Zyxel | 'racoon' | 45 |
| Planet | 'pppd245' | 2 | Zyxel | 'fadd' | 40 |
| Planet | 'cwmpd' | 2 | Zyxel | 'tr69' | 39 |
| Planet | 'prog.cgi' | 2 | Zyxel | 'vpppd' | 39 |
| Planet | 'pppdo' | 2 | Zyxel | 'radiusc' | 34 |
| Planet | 'sendReport' | 2 | Zyxel | 'trace' | 32 |
| Planet | 'dccupdate' | 2 | Zyxel | 'httpd' | 32 |
| Planet | 'HA' | 2 | Zyxel | 'zhttpd' | 29 |
| Planet | 'htmldoc' | 2 | Zyxel | 'zyecho_client' | 26 |
| Planet | 'pppoecd' | 2 | Zyxel | 'zyxel_xmpp_client' | 13 |
| Planet | 'SystemServer' | 2 | Zyxel | 'zapiBLEService' | 13 |
| Planet | 'boa' | 2 | Zyxel | 'pdbtool' | 9 |
| Planet | 'hi3518' | 2 | Zyxel | 'sipclient' | 8 |
| Planet | 'panod' | 2 | Zyxel | 'zytr069main' | 6 |
| Planet | 'aistreamer' | 2 | Zyxel | 'bgpd' | 6 |
| Planet | 'badblocks' | 1 | Zyxel | 'dma' | 6 |
| Planet | 'dma' | 1 | Zyxel | 'sippxy.elf' | 5 |
| Planet | 'freshsnort' | 1 | Zyxel | 'wmgeniesrv' | 5 |
| Planet | 'p3scan' | 1 | Zyxel | 'hlasd' | 5 |
| Planet | 'perl' | 1 | Zyxel | 'squid' | 5 |
| QNAP | 'slapd-bind' | 23 | Zyxel | 'tools_mpt.cgi' | 5 |
| QNAP | 'mount.davfs' | 23 | Zyxel | 'snmpd' | 4 |
| QNAP | 'utilRequest.cgi' | 22 | Zyxel | 'cli' | 3 |
| QNAP | 'badblocks' | 11 | Zyxel | 'pppd_3g' | 3 |
| QNAP | '638C43' | 1 | Zyxel | 'wsccmd' | 3 |
| QNAP | '6C6CF5' | 1 | Zyxel | 'zySAS' | 2 |
| QNAP | '69F2D0' | 1 | Zyxel | 'cfm' | 2 |
| QNAP | '6B221E' | 1 | Zyxel | 'lte_srv_diag' | 2 |
| Rotek | 'radvd' | 1 | Zyxel | 'smart-polling-service' | 2 |
| Synology | 'hostapd' | 141 | Zyxel | 'boa' | 2 |
| Synology | 'nsupdate' | 135 | Zyxel | 'ag' | 2 |
| Synology | 'synorelayd' | 131 | Zyxel | 'sysd' | 2 |
| Synology | 'synosearchagent' | 130 | Zyxel | 'auto-ip' | 1 |
| Synology | 'share-hook' | 114 | Zyxel | 'ripngd' | 1 |
| Synology | 'findhostd' | 113 | Zyxel | 'uplink_qos' | 1 |
| Synology | 'volume-hook' | 112 | Zyxel | 'wireless.cgi' | 1 |

Table A.56: Violated Binaries discovered for Pseudo Random Number Generators (*PRNGs*) **R1** and **R2**, entry and not discovered $\phi$ case.

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| ***K1*** - Constant Passwords on a KDF/PBE | | | ***K4*** - 'Weak' underlying hash function on a KDF/PBE | | |
| ASUS | 'rc' | 4 | ASUS | 'rc' | 6 |
| ASUS | 'qcmap_auth' | 2 | ASUS | 'login.shadow' | 2 |
| Arris | 'sc_zipen' | 4 | Actiontec | 'cm_logic' | 1 |
| D-Link | 'smm' | 121 | Arris | 'uhttpd' | 3 |
| D-Link | 'commander' | 9 | Belkin | 'cfm' | 1 |
| D-Link | 'admin.cgi' | 3 | D-Link | 'sslvpnConfig' | 122 |
| LinkSys | 'eurl' | 45 | D-Link | 'smm' | 121 |
| LinkSys | 'main_bin' | 2 | D-Link | 'admin.cgi' | 59 |
| NETGEAR | 'smm' | 27 | D-Link | 'tinylogin' | 36 |
| NETGEAR | 'cli' | 1 | D-Link | 'cm_logic' | 18 |
| QNAP | 'change_password.cgi' | 23 | D-Link | 'logic' | 4 |
| QNAP | 'authLogin.cgi' | 6 | D-Link | 'uhttpd' | 4 |
| QNAP | '6EC27A' | 1 | D-Link | 'busybox' | 2 |
| QNAP | '6D1AB2' | 1 | D-Link | 'cmd' | 2 |
| QNAP | '638C43' | 1 | D-Link | 'rc' | 2 |
| QNAP | '6C6CF5' | 1 | D-Link | 'tssa' | 1 |

| | | | | | |
|---|---|---|---|---|---|
| QNAP | '69F2D0' | 1 | EdiMax | 'cfg_manager' | 5 |
| QNAP | '6B221E' | 1 | EdiMax | 'ftpd' | 5 |
| TP-Link | 'qcmap_auth' | 34 | EdiMax | 'boa' | 4 |
| TP-Link | 'pure-pw' | 15 | EdiMax | 'startup' | 4 |
| Trendnet | 'daemon_fsp_app' | 16 | Inmarsat | 'unix_chkpwd' | 3 |
| Trendnet | 'goahead' | 2 | Inmarsat | 'unix_update' | 3 |
| Trendnet | 'cwsysd' | 1 | LinkSys | 'eurl' | 45 |
| Trendnet | 'main.cgi' | 1 | LinkSys | 'main_bin' | 2 |
| Trendnet | 'accountd' | 1 | LinkSys | 'uhttpd' | 1 |
| Zyxel | 'zcmd' | 1 | LinkSys | 'boa' | 1 |
| **K2** - Constant salt or no salts on a KDF/PBE | | | NETGEAR | 'unix_chkpwd' | 4213 |
| ASUS | 'rc' | 6 | NETGEAR | 'unix_update' | 4213 |
| ASUS | 'login.shadow' | 2 | NETGEAR | 'uhttpd' | 141 |
| Actiontec | 'cm_logic' | 1 | NETGEAR | 'busybox' | 134 |
| Arris | 'uhttpd' | 3 | NETGEAR | 'rc' | 61 |
| Belkin | 'cfm' | 1 | NETGEAR | 'appliance_mgr_cli' | 27 |
| D-Link | 'sslvpnConfig' | 122 | NETGEAR | 'smm' | 27 |
| D-Link | 'smm' | 121 | NETGEAR | 'ngadmin.cgi' | 21 |
| D-Link | 'tinylogin' | 36 | NETGEAR | 'httpd' | 17 |
| D-Link | 'cm_logic' | 18 | NETGEAR | 'pam_extrausers_update' | 16 |
| D-Link | 'logic' | 4 | NETGEAR | 'pam_extrausers_chkpwd' | 16 |
| D-Link | 'uhttpd' | 4 | NETGEAR | 'cm_logic' | 5 |
| D-Link | 'busybox' | 2 | NETGEAR | 'lc_up' | 2 |
| D-Link | 'cmd' | 2 | NETGEAR | 'mongoose' | 1 |
| D-Link | 'rc' | 2 | NETGEAR | 'password_crypt' | 1 |
| D-Link | 'tssa' | 1 | Netis | 'startup' | 2 |
| EdiMax | 'cfg_manager' | 5 | Netis | 'boa' | 2 |
| EdiMax | 'ftpd' | 5 | Planet | 'uhttpd' | 49 |
| EdiMax | 'boa' | 4 | Planet | 'cgiMain' | 8 |
| EdiMax | 'startup' | 4 | Planet | 'login' | 4 |
| LinkSys | 'eurl' | 45 | Planet | 'unix_update' | 4 |
| LinkSys | 'main_bin' | 2 | Planet | 'unix_chkpwd' | 4 |
| LinkSys | 'uhttpd' | 1 | Planet | 'swctrl' | 3 |
| LinkSys | 'boa' | 1 | Planet | 'httpd' | 2 |
| NETGEAR | 'uhttpd' | 141 | Planet | 'startup' | 1 |
| NETGEAR | 'busybox' | 134 | QNAP | '6EC27A' | 1 |
| NETGEAR | 'rc' | 61 | QNAP | '6D1AB2' | 1 |
| NETGEAR | 'appliance_mgr_cli' | 27 | QNAP | '638C43' | 1 |
| NETGEAR | 'smm' | 27 | QNAP | '6C6CF5' | 1 |
| NETGEAR | 'httpd' | 17 | QNAP | '69F2D0' | 1 |
| NETGEAR | 'cm_logic' | 5 | QNAP | '6B221E' | 1 |
| NETGEAR | 'password_crypt' | 1 | Synology | 'unix_chkpwd' | 176 |
| Netis | 'startup' | 2 | Synology | 'getty' | 128 |
| Netis | 'boa' | 2 | Synology | 'synouser' | 43 |
| Planet | 'uhttpd' | 49 | Synology | 'afpd' | 38 |
| Planet | 'cgiMain' | 8 | Synology | 'synorcvol' | 37 |
| Planet | 'login' | 4 | Synology | 'rsrcmonitor.cgi' | 36 |
| Planet | 'swctrl' | 3 | Synology | 'manutild' | 7 |
| Planet | 'httpd' | 2 | Synology | 'synocheckshare' | 7 |
| Planet | 'startup' | 1 | Synology | 'scemd' | 5 |
| QNAP | '6EC27A' | 1 | Synology | 'findhostd' | 2 |
| QNAP | '6D1AB2' | 1 | Synology | 'syno-cloud-syncd' | 2 |
| QNAP | '638C43' | 1 | Synology | 'cloud-sync-encrypt-tool' | 2 |
| QNAP | '6C6CF5' | 1 | Synology | 'cloud-sync-starter' | 2 |
| QNAP | '69F2D0' | 1 | Synology | 'mysqld' | 1 |
| QNAP | '6B221E' | 1 | TP-Link | 'uhttpd' | 675 |
| Synology | 'synorcvol' | 37 | TP-Link | 'pure-pw' | 68 |
| TP-Link | 'uhttpd' | 675 | TP-Link | 'login.shadow' | 33 |
| TP-Link | 'login.shadow' | 33 | TP-Link | 'mysqld' | 1 |
| TP-Link | 'chsh.shadow' | 15 | TP-Link | 'mariabackup' | 1 |
| TP-Link | 'passwd.shadow' | 15 | Tenda | 'uhttpd' | 4 |
| TP-Link | 'gpasswd' | 15 | Thuraya | 'uhttpd' | 2 |
| TP-Link | 'su' | 15 | Totolink | 'uhttpd' | 1 |
| TP-Link | 'newusers' | 15 | Trendnet | 'uhttpd' | 26 |
| TP-Link | 'chgpasswd' | 15 | Trendnet | 'daemon_fsp_app' | 16 |
| TP-Link | 'chpasswd.shadow' | 15 | Trendnet | 'startup' | 3 |
| TP-Link | 'chfn.shadow' | 15 | Trendnet | 'boa' | 3 |

| | | | | | |
|---|---|---|---|---|---|
| TP-Link | 'pure-pw' | 15 | Trendnet | 'busybox' | 2 |
| Tenda | 'uhttpd' | 4 | Trendnet | 'goahead' | 2 |
| Thuraya | 'uhttpd' | 2 | Trendnet | 'init' | 1 |
| Totolink | 'uhttpd' | 1 | Trendnet | 'logic' | 1 |
| Trendnet | 'uhttpd' | 26 | Trendnet | 'cwsysd' | 1 |
| Trendnet | 'daemon_fsp_app' | 16 | Ubiquiti | 'unix_chkpwd' | 354 |
| Trendnet | 'startup' | 3 | Ubiquiti | 'unix_update' | 343 |
| Trendnet | 'boa' | 3 | Ubiquiti | 'uhttpd' | 78 |
| Trendnet | 'busybox' | 2 | Western-Digital | 'unix_chkpwd' | 5 |
| Trendnet | 'goahead' | 2 | Western-Digital | 'unix_update' | 5 |
| Trendnet | 'init' | 1 | Xiaomi | 'uhttpd' | 299 |
| Trendnet | 'logic' | 1 | Zyxel | 'unix_update' | 80 |
| Trendnet | 'cwsysd' | 1 | Zyxel | 'unix_chkpwd' | 80 |
| Ubiquiti | 'uhttpd' | 78 | Zyxel | 'makepwd' | 66 |
| Xiaomi | 'uhttpd' | 299 | Zyxel | 'uhttpd' | 36 |
| Zyxel | 'makepwd' | 66 | Zyxel | 'mini_httpd' | 13 |
| Zyxel | 'uhttpd' | 36 | Zyxel | 'cfg_manager' | 3 |
| Zyxel | 'mini_httpd' | 13 | Zyxel | 'boa' | 2 |
| Zyxel | 'cfg_manager' | 3 | Zyxel | 'startup' | 2 |
| Zyxel | 'boa' | 2 | Zyxel | 'wireless.cgi' | 1 |
| Zyxel | 'startup' | 2 | - | - | - |
| Zyxel | 'wireless.cgi' | 1 | - | - | - |
| ***K3*** - 'Weak' number of iteration on a KDF/PBE | | | - | - | - |
| Arris | 'sc_zipen' | 4 | - | - | - |
| DrayTek | 'fw_printenv' | 62 | - | - | - |
| LinkSys | 'eurl' | 45 | - | - | - |
| NETGEAR | 'lc_up' | 2 | - | - | - |
| NETGEAR | 'mongoose' | 1 | - | - | - |
| Synology | 'cloud-sync-starter' | 2 | - | - | - |
| Synology | 'mysqld' | 1 | - | - | - |
| TP-Link | 'mysqld' | 1 | - | - | - |
| TP-Link | 'mariabackup' | 1 | - | - | - |
| Trendnet | 'daemon_fsp_app' | 16 | - | - | - |
| Ubiquiti | 'ubntbox' | 76 | - | - | - |
| Xiaomi | 'ss-local' | 68 | - | - | - |
| Xiaomi | 'ss-redir' | 54 | - | - | - |
| Xiaomi | 'ss-tunnel' | 3 | - | - | - |
| Zyxel | 'zycfgfilter' | 90 | - | - | - |
| Zyxel | 'zcmd' | 74 | - | - | - |

Table A.57: Violated Binaries discovered for Key Derivation Functions (*KDFs*) and Password Based Encryption (*PBE*) rules **K1**, **K2**, **K3** and **K4**, entry and not discovered $\phi$ case.

| Vendor | Binary Name | # firmwares | Vendor | Binary Name | # firmwares |
|---|---|---|---|---|---|
| ***M3*** - Non-secure key length on a MAC function | | | ***M1*** - Constant Encryption/Decryption Keys on a MAC | | |
| ASUS | 'hostapd' | 29 | DrayTek | 'tr069_client' | 93 |
| Alfa | 'wpad' | 8 | NETGEAR | 'Netgear_ddns' | 144 |
| LinkSys | 'hostapd' | 1 | NETGEAR | 'httpd' | 24 |
| NETGEAR | 'dimclient' | 31 | NETGEAR | 'ntgrddns' | 14 |
| NETGEAR | 'hostapd' | 1 | - | - | - |
| Synology | 'hostapd' | 1 | - | - | - |
| TP-Link | 'hostapd' | 7 | - | - | - |
| Tenda | 'hostapd' | 1 | - | - | - |
| ***M2*** - 'Weak' underlying hash function on a MAC | | | | | |
| ASUS | 'wpa_supplicant' | 151 | Planet | 'snmpd' | 50 |
| ASUS | 'hostapd' | 95 | Planet | 'wpa_supplicant' | 27 |
| ASUS | 'wpa_supplicant-2.7' | 29 | Planet | 'prog.cgi' | 2 |
| ASUS | 'racoon' | 2 | Planet | 'boa' | 2 |
| AVM | 'hostapd' | 5 | Planet | 'xsupplicant' | 1 |
| AVM | 'wpa_supplicant' | 5 | QNAP | 'wpa_supplicant' | 13 |
| Actiontec | 'racoon' | 2 | Synology | 'wpa_supplicant' | 160 |
| Alfa | 'wpad' | 33 | Synology | 'hostapd' | 145 |
| D-Link | 'prog-cgi' | 126 | Synology | 'git-imap-send' | 4 |
| D-Link | 'mdb' | 60 | TP-Link | 'wpa_supplicant' | 50 |
| D-Link | 'wpa_supplicant' | 23 | TP-Link | 'racoon' | 47 |

| | | | | | |
|---|---|---|---|---|---|
| D-Link | 'snmptrap' | 7 | TP-Link | 'hostapd' | 28 |
| D-Link | 'snmpd' | 7 | Tenda | 'racoon' | 22 |
| D-Link | 'hostapd' | 7 | Tenda | 'hostapd' | 2 |
| D-Link | 'dam' | 4 | Tenda | 'wpa_supplicant' | 1 |
| D-Link | 'racoon' | 4 | Tenvis | 'onvif' | 1 |
| D-Link | 'xsupplicant' | 4 | Totolink | 'wpa_supplicant' | 4 |
| D-Link | 'prog.cgi' | 3 | Trendnet | 'wpa_supplicant' | 18 |
| DrayTek | 'hostapd' | 5 | Trendnet | 'hostapd' | 2 |
| DrayTek | 'wpa_supplicant' | 5 | Ubiquiti | 'wpad' | 264 |
| EdiMax | 'device_service' | 5 | Ubiquiti | 'hostapd' | 39 |
| EdiMax | 'racoon' | 3 | Ubiquiti | 'wpa_supplicant' | 37 |
| EdiMax | 'wpa_supplicant' | 2 | Western-Digital | 'wpa_supplicant' | 4 |
| EdiMax | 'hostapd' | 2 | Western-Digital | 'hostapd' | 3 |
| Inmarsat | 'hostapd' | 3 | Xiaomi | 'wpa_supplicant' | 13 |
| LinkSys | 'dhclient' | 71 | Xiaomi | '245506E' | 1 |
| LinkSys | 'wpa_supplicant' | 11 | Zyxel | 'racoon' | 21 |
| LinkSys | 'hostapd' | 9 | Zyxel | 'radclient' | 15 |
| LinkSys | 'onvif1.0' | 4 | Zyxel | 'radeapclient' | 15 |
| LinkSys | 'onvif2' | 4 | Zyxel | 'radiusd' | 6 |
| MicroTik | 'ipsec' | 350 | Zyxel | 'hostapd' | 1 |
| NETGEAR | 'wpa_supplicant' | 69 | Zyxel | 'wpa_supplicant-macsec' | 1 |
| NETGEAR | 'hostapd' | 43 | Zyxel | 'wpa_supplicant' | 1 |
| NETGEAR | 'wpa_supplicant-macsec' | 1 | - | - | - |

Table A.58: Violated Binaries discovered for Message Authentication Codes (*MACs*) rules **M1**, **M2** and **M3**, entry and not discovered $\phi$ case.

# B

# Appendix - Supported Cryptographic Primitives

## B.1. Symmetric Key Cryptography

For Symmetric Key Cryptography, the analysis can discover the following block ciphers, stream ciphers, and mode of operations.

Block ciphers list:

1. AES
2. DES
3. Two-key TDEA (triple DES)
4. Three-key TDEA (triple DES)
5. Blowfish
6. RC2
7. Camellia
8. CAST
9. CAST5
10. IDEA
11. TWOFISH
12. SERPENT
13. SAFER SK

Stream ciphers list:

1. RC4
2. RC5
3. CHACHA20

Modes of Operation:

1. ECB
2. CBC
3. OFB
4. CFB
5. CTR
6. XTS
7. CFB1
8. CFB8
9. CFB64
10. CFB128
11. OFB8
12. OFB64
13. OFB128

In addition, the analysis can identify:

- Key sizes. For instance, AES uses three key sizes 128, 192 or 256 bits.
- IV size when applicable.
- Direction: Encryption or decryption.

## B.2. Public Key Cryptography

For Public Key Cryptography, the analysis can cover the following list of algorithms:

1. RSA
2. X.509 standard
3. Digital Signatures with various combinations, e.g. DSA, ECDSA

Additionally, the analysis can discover:

- RSA padding schemes
- RSA key sizes
- Underlying hash functions for X.509 and digital signatures.
- Direction: Encryption or decryption (for public key encryption)

## B.3. Pseudo Random Number Generators (PRNGs)

For Pseudo Random Number Generators (PRNGs), the analysis can discover the following list of algorithms:

1. dev/urandom, dev/random, getpid(), time(), provided as seed to rand functions.
2. OpenSSL rand functions.
3. Libc rand functions.
4. GnuPG, libgcrypt rand functions.

## B.4. Cryptographic One-way Hash functions

For Cryptographic One-way Hash functions, the analysis can identify the following list of algorithms:

1. MD2
2. MD4
3. MD5
4. MDC2
5. SHA
6. SHA1
7. SHA224
8. SHA256
9. SHA384
10. SHA512
11. RIPEMD160
12. SHA3-224
13. SHA3-256
14. SHA3-384
15. SHA3-512
16. BLAKE2B
17. BLAKE2S
18. SHAKE128
19. SHAKE256
20. BLAKE2B-512
21. BLAKE2B-384
22. BLAKE2B-256
23. BLAKE2B-160
24. BLAKE2S-256
25. BLAKE2S-224
26. BLAKE2S-160
27. BLAKE2S-128

## B.5. Key Derivation Functions (KDFs) and Password Hashes

For Key Derivation Functions (KDFs) and Password Hashes, the following algorithms are covered:

1. HMAC KDF
2. BCRYPT
3. PBKDF1
4. PBKDF2
5. SCRYPT
6. crypt (Linux)

Furthermore, the analysis can discover:

- Underlying hash functions where applicable
- Iterations

## B.6. Message Authentication Codes (MACs)

For Message Authentication Codes (MACs), the following list of MACs is discovered:

1. HMAC
2. BLAKE2 keyed hash

In addition, the analysis can identify:

- Underlying hash functions where applicable
- Key size

## B.7. Authenticated Encryption with associated data

For Authenticated Encryption with associated data, the following algorithms are covered:

1. AES
2. CHACHA20 (stream)

The following Modes of Operations for authenticated encryption are discovered:

1. CCM (CBC-MAC)
2. GCM
3. POLY1305
4. OCB

Additionally, the analysis can discover:

- Key size
- IV size
- Additional Authenticated Data and Tag

# Bibliography


[1] Angr. URL https://github.com/angr/angr.

[2] Binary analysis next generation (bang). URL https://github.com/armijnhemel/binaryanalysis-ng.

[3] Crypto++. URL https://cryptopp.com/.

[4] firmwalker. URL https://github.com/craigz28/firmwalker.

[5] The GNU C library. URL https://www.gnu.org/software/libc/.

[6] GNU Privacy Guard, . URL https://gnupg.org/.

[7] GnuTLS, . URL https://gnutls.org/.

[8] hashcat, advanced password recovery. URL https://hashcat.net/hashcat/.

[9] Sodium, . URL https://libsodium.org.

[10] LibTomCrypt, . URL https://www.libtom.net/LibTomCrypt/.

[11] mbed TLS . URL https://tls.mbed.org/.

[12] libmcrypt - encryption/decryption library. URL https://linux.die.net/man/3/mcrypt.

[13] Nettle - a low-level cryptographic library. URL https://www.lysator.liu.se/~nisse/nettle/.

[14] OpenSSL, Cryptography and SSL/TLS Toolkit. URL https://www.openssl.org/.

[15] QEMU. URL https://www.qemu.org/.

[16] uClibc-ng - embedded c library. URL https://uclibc-ng.org/.

[17] wolfSSL embedded tls library. URL https://www.wolfssl.com/.

[18] PKCS #1: RSA Encryption Version 1.5. RFC 2313, March 1998.

[19] PKCS #1: RSA Cryptography Specifications Version 2.0. RFC 2437, October 1998.

[20] Prohibiting RC4 Cipher Suites. RFC 7465, February 2015.

[21] PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, November 2016.

[22] PKCS #5: Password-Based Cryptography Specification Version 2.1. RFC 8018, January 2017.

[23] Iso/iec/ieee international standard - systems and software engineering–vocabulary. *ISO/IEC/IEEE 24765:2017(E)*, pages 1–541, 2017.

[24] Iot: a malware story, 2019. URL https://securelist.com/iot-a-malware-story/94451/.

[25] Breaking the d-link dir3060 firmware encryption, 2020. URL https://0x00sec.org/t/breaking-the-d-link-dir3060-firmware-encryption-recon-part-1.

[26] Blazej Adamczyk. Multiple vulnerabilities in all versions of asus routers, 2017. URL https://seclists.org/fulldisclosure/2018/Jan/63.

[27] Sharmin Afrose, Sazzadur Rahaman, and Danfeng Yao. Cryptoapi-bench: A comprehensive benchmark on java cryptographic API misuses. In *2019 IEEE Cybersecurity Development, SecDev 2019, Tysons Corner, VA, USA, September 23-25, 2019*, pages 49–61. IEEE, 2019. doi: 10.1109/SecDev.2019.00017. URL https://doi.org/10.1109/SecDev.2019.00017.

[28] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1093–1110. USENIX Association, 2017. URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[29] Xiaoyun Wangm Benne de Weger Arjen Lenstra. Colliding x.509 certificates based on md5-collisions. URL https://www.win.tue.nl/~bdeweger/CollidingCertificates/.

[30] Gregory Basior. 6 new vulnerabilities found on d-link home routers, 2020. URL https://unit42.paloaltonetworks.com/6-new-d-link-vulnerabilities-found-on-home-routers/.

[31] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998. doi: 10.1007/BFb0055716. URL https://doi.org/10.1007/BFb0055716.

[32] Alexei Bulazel. Working With Ghidra's P-Code To Identify Vulnerable Function Calls, 2019. URL https://www.riverloopsecurity.com/blog/2019/05/pcode/.

[33] Alexia Chatzikonstantinou, Christoforos Ntantogian, Georgios Karopoulos, and Christos Xenakis. Evaluation of cryptography usage in android applications. *EAI Endorsed Trans. Security Safety*, 3(9):e4, 2016. doi: 10.4108/eai.3-12-2015.2262471. URL https://doi.org/10.4108/eai.3-12-2015.2262471.

[34] Daming D. Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. URL http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/towards-automated-dynamic-analysis-linux-based-embedded-firmware.pdf.

[35] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. URL http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_01A-1_Chen_paper.pdf.

[36] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 95–110. USENIX Association, 2014. URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin.

[37] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. Automated dynamic firmware analysis at scale: A case study on embedded web interfaces. In Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang, editors, *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, pages 437–448. ACM, 2016. doi: 10.1145/2897845.2897900. URL https://doi.org/10.1145/2897845.2897900.

[38] Yaniv David, Nimrod Partush, and Eran Yahav. Firmup: Precise static detection of common vulnerabilities in firmware. In Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar, editors, *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, pages 392–404. ACM, 2018. doi: 10.1145/3173162.3177157. URL https://doi.org/10.1145/3173162.3177157.

[39] Chris Eagle and Kara Nance. *The Ghidra Book: The Definitive Guide*. No Starch Press, 2020.

[40] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 73–84. ACM, 2013. doi: 10.1145/2508859.2516693. URL https://doi.org/10.1145/2508859.2516693.

[41] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 73–84. ACM, 2013. doi: 10.1145/2508859.2516693. URL https://doi.org/10.1145/2508859.2516693.

[42] Sebastian Eschweiler, Khaled Yakdan, and Elmar Gerhards-Padilla. discovre: Efficient cross-architecture identification of bugs in binary code. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. URL http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/discovre-efficient-cross-architecture-identification-bugs-binary-code.pdf.

[43] Johannes Feichtner, David Missmann, and Raphael Spreitzer. Automated binary analysis on ios: A case study on cryptographic misuse in ios applications. In Panos Papadimitratos, Kevin R. B. Butler, and Christina Pöpper, editors, *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WiSec 2018, Stockholm, Sweden, June 18-20, 2018*, pages 236–247. ACM, 2018. doi: 10.1145/3212480.3212487. URL https://doi.org/10.1145/3212480.3212487.

[44] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. Scalable graph-based bug search for firmware images. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 480–491. ACM, 2016. doi: 10.1145/2976749.2978370. URL https://doi.org/10.1145/2976749.2978370.

[45] Fraunhofer FKIE. Firmware analysis and comparison tool (fact), . URL https://github.com/fkie-cad/FACT_core.

[46] Fraunhofer FKIE. Firmware analysis and comparison tool (fact) extractor, . URL https://github.com/fkie-cad/fact_extractor.

[47] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2001. doi: 10.1007/3-540-45537-X_1. URL https://doi.org/10.1007/3-540-45537-X_1.

[48] Free Standards Group. Interfaces for libcrypt. URL https://refspecs.linuxbase.org/LSB_3.0.0/LSB-PDA/LSB-PDA/libcrypt.html.

[49] Hex-Rays. The IDA Pro Disassembler and Debugger. URL https://www.hex-rays.com/products/ida/.

[50] Heike Hofmann, Karen Kafadar, and Hadley Wickham. Letter-value plots: Boxplots for large data. Technical report, had.co.nz, 2011.

[51] Intel. Cve binary tool. URL https://github.com/intel/cve-bin-tool.

[52] Antoine Joux. Authentication Failures in NIST version of GCM. National Institute of Standards and Technology (NIST). URL https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/800-38-series-drafts/gcm/joux_comments.pdf.

[53] Jesse D. Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digit. Investig.*, 3(Supplement-1):91–97, 2006. doi: 10.1016/j.diin.2006.06.015. URL https://doi.org/10.1016/j.diin.2006.06.015.

[54] ReFirm Labs. Binwalk. URL https://github.com/ReFirmLabs/binwalk.

[55] David Lazar, Haogang Chen, Xi Wang, and Nickolai Zeldovich. Why does cryptographic software fail?: a case study and open problems. In *Asia-Pacific Workshop on Systems, APSys'14, Beijing, China, June 25-26, 2014*, pages 7:1–7:7. ACM, 2014. doi: 10.1145/2637166.2637237. URL https://doi.org/10.1145/2637166.2637237.

[56] David Lazar, Haogang Chen, Xi Wang, and Nickolai Zeldovich. Why does cryptographic software fail?: a case study and open problems. In *Asia-Pacific Workshop on Systems, APSys'14, Beijing, China, June 25-26, 2014*, pages 7:1–7:7. ACM, 2014. doi: 10.1145/2637166.2637237. URL https://doi.org/10.1145/2637166.2637237.

[57] Arjen K. Lenstra and Benne de Weger. On the possibility of constructing meaningful hash collisions for public keys. In Colin Boyd and Juan Manuel González Nieto, editors, *Information Security and Privacy, 10th Australasian Conference, ACISP 2005, Brisbane, Australia, July 4-6, 2005, Proceedings*, volume 3574 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 2005. doi: 10.1007/11506157_23. URL https://doi.org/10.1007/11506157_23.

[58] Gaëtan Leurent and Thomas Peyrin. From collisions to chosen-prefix collisions application to full SHA-1. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 527–555. Springer, 2019. doi: 10.1007/978-3-030-17659-4_18. URL https://doi.org/10.1007/978-3-030-17659-4_18.

[59] Gaëtan Leurent and Thomas Peyrin. SHA-1 is a shambles: First chosen-prefix collision on SHA-1 and application to the PGP web of trust. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1839–1856. USENIX Association, 2020. URL https://www.usenix.org/conference/usenixsecurity20/presentation/leurent.

[60] Hikohiro Y Li and Yuki Osawa. Understanding the iot threat landscape and a home appliance manufacturer's approach to counter threats to iot, 2019. URL https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/.

[61] Yong Li, Yuanyuan Zhang, Juanru Li, and Dawu Gu. icryptotracer: Dynamic analysis on misuse of cryptography functions in ios applications. In Man Ho Au, Barbara Carminati, and C.-C. Jay Kuo, editors, *Network and System Security - 8th International Conference, NSS 2014, Xi'an, China, October 15-17, 2014, Proceedings*, volume 8792 of *Lecture Notes in Computer Science*, pages 349–362. Springer, 2014. doi: 10.1007/978-3-319-11698-3_27. URL https://doi.org/10.1007/978-3-319-11698-3_27.

[62] Chi-Keung Luk, Robert S. Cohn, Robert Muth, Harish Patil, Artur Klauser, P. Geoffrey Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim M. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In Vivek Sarkar and Mary W. Hall, editors, *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005*, pages 190–200. ACM, 2005. doi: 10.1145/1065010.1065034. URL https://doi.org/10.1145/1065010.1065034.

[63] Dave McDaniel. Talos-2021-1283 || cisco talos intelligence group - comprehensive threat intelligence, 2021. URL https://talosintelligence.com/vulnerability_reports/TALOS-2021-1283.

[64] Bodo Moeller. Security of cbc ciphersuites in ssl/tls: Problems and countermeasures. URL https://www.openssl.org/~bodo/tls-cbc.txt.

[65] Chao Mu, Ming Yang, Zhenya Chen, and Biao Wang. Research on RSA padding identification method in IoT firmwares. *Journal of Physics: Conference Series*, 1570:012061, jun 2020. doi: 10.1088/1742-6596/1570/1/012061. URL https://doi.org/10.1088%2F1742-6596%2F1570%2F1%2F012061.

[66] Marius Muench, Jan Stijohann, Frank Kargl, Aurélien Francillon, and Davide Balzarotti. What you corrupt is not what you crash: Challenges in fuzzing embedded devices. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. URL http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_01A-4_Muench_paper.pdf.

[67] Ildar Muslukhov, Yazan Boshmaf, and Konstantin Beznosov. Source attribution of cryptographic API misuse in android applications. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 133–146. ACM, 2018. doi: 10.1145/3196494.3196538. URL https://doi.org/10.1145/3196494.3196538.

[68] Asuka Nakajima. Hunting vulnerable oem iot devices at scale, 2019. URL https://i.blackhat.com/eu-19/Thursday/eu-19-Nakajima-OEM-Finder-Hunting-Vulnerable-OEM-IoT-Devices-At-Scale-2.pdf.

[69] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In Jeanne Ferrante and Kathryn S. McKinley, editors, *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007*, pages 89–100. ACM, 2007. doi: 10.1145/1250734.1250746. URL https://doi.org/10.1145/1250734.1250746.

[70] National Security Agency (NSA). Ghidra software reverse engineering framework. URL https://ghidra-sre.org/.

[71] National Institute of Standards and Technology (NIST). Transitions: Recommendation fro transitioning the use of cryptographic algortithms and key lengths, . URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf.

[72] National Institute of Standards and Technology (NIST). Recommendation for password-based key derivation, . URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf.

[73] National Institute of Standards and Technology (NIST). Common platform enumeration (cpe) dictionary, . URL https://nvd.nist.gov/products/cpe.

[74] National Institute of Standards and Technology (NIST). National vulnerability database (nvd), . URL https://nvd.nist.gov/vuln/data-feeds.

[75] National Institute of Standards and Technology (NIST). Federal information processing standards (FIPS) 197, Advanced Encryption Standard (AES). 2001.

[76] National Institute of Standards and Technology (NIST). 800-38a, recommendation for block cipher modes of operation: Methods and techniques, 2001. URL https://doi.org/10.6028/NIST.SP.800-38A.

[77] National Institute of Standards and Technology (NIST). Federal information processing standards (FIPS) 180-4, Secure Hash Standard. 2015.

[78] Open Web Application Security Project (OWASP). Embedded application security best practices, . URL https://scriptingxss.gitbook.io/embedded-appsec-best-practices/.

[79] Open Web Application Security Project (OWASP). C-based toolchain hardening, . URL https://wiki.owasp.org/index.php/C-Based_Toolchain_Hardening.

[80] Open Web Application Security Project (OWASP). Testing for weak encryption, . URL https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/04-Testing_for_Weak_Encryption.

[81] Open Web Application Security Project (OWASP). Internet of things top 10, 2018. URL https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf.

[82] Jannik Pewny, Behrad Garmany, Robert Gawlik, Christian Rossow, and Thorsten Holz. Cross-architecture bug search in binary executables. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 709–724. IEEE Computer Society, 2015. doi: 10.1109/SP.2015.49. URL https://doi.org/10.1109/SP.2015.49.

[83] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng (Daphne) Yao. Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2455–2472. ACM, 2019. doi: 10.1145/3319535.3345659. URL https://doi.org/10.1145/3319535.3345659.

[84] Nilo Redini, Aravind Machiry, Ruoyu Wang, Chad Spensky, Andrea Continella, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Karonte: Detecting insecure multi-binary interactions in embedded firmware. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1544–1561. IEEE, 2020. doi: 10.1109/SP40000.2020.00036. URL https://doi.org/10.1109/SP40000.2020.00036.

[85] Pedro Ribeiro and Radek Domanski. Exploiting the tp-link archer a7 at pwn2own tokyo, 2020. URL https://www.zerodayinitiative.com/blog/2020/4/6/exploiting-the-tp-link-archer-c7-at-pwn2own-tokyo.

[86] Scrapinghub. Scrapy. URL https://scrapy.org/.

[87] Shuai Shao, Guowei Dong, Tao Guo, Tianchang Yang, and Chenjie Shi. Modelling analysis and auto-detection of cryptographic misuse in android applications. In *IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, DASC 2014, Dalian, China, August 24-27, 2014*, pages 75–80. IEEE Computer Society, 2014. doi: 10.1109/DASC.2014.22. URL https://doi.org/10.1109/DASC.2014.22.

[88] Tohid Shekari and Raheem Beyah. Iot skimmer: Energy market manipulation through high-wattage iot botnets, 2020. URL https://i.blackhat.com/USA-20/Wednesday/us-20-Shekari-IoT-Skimmer-Energy-Market-Manipulation-Through-High-Wattage-IoT-Botnets.pdf.

[89] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. URL https://www.ndss-symposium.org/ndss2015/firmalice-automatic-detection-authentication-bypass-vulnerabilities-binary-firmware.

[90] Prashast Srivastava, Hui Peng, Jiahao Li, Hamed Okhravi, Howard E. Shrobe, and Mathias Payer. Firmfuzz: Automated iot firmware introspection and analysis. In Peng Liu and Yuqing Zhang, editors, *Proceedings of the 2nd International ACM Workshop on Security and Privacy for*

*the Internet-of-Things, IoT S&P@CCS 2019, London, UK, November 15, 2019*, pages 15–21. ACM, 2019. doi: 10.1145/3338507.3358616. URL https://doi.org/10.1145/3338507.3358616.

[91] Marc Stevens, Arjen Lenstra, and Benne de Weger. Chosen-prefix collisions for md5 and colliding x.509 certificates for different identities. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, pages 1–22, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-72540-4.

[92] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017. doi: 10.1007/978-3-319-63688-7_19. URL https://doi.org/10.1007/978-3-319-63688-7_19.

[93] MIT Kerberos Team. Kerberos 5. URL https://web.mit.edu/kerberos/krb5-latest/.

[94] VirusTotal. YARA. URL https://github.com/virustotal/yara.

[95] Qing Wang, Juanru Li, Yuanyuan Zhang, Hui Wang, Yikun Hu, Bodong Li, and Dawu Gu. Nativespeaker: Identifying crypto misuses in android native code libraries. In Xiaofeng Chen, Dongdai Lin, and Moti Yung, editors, *Information Security and Cryptology - 13th International Conference, Inscrypt 2017, Xi'an, China, November 3-5, 2017, Revised Selected Papers*, volume 10726 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2017. doi: 10.1007/978-3-319-75160-3_19. URL https://doi.org/10.1007/978-3-319-75160-3_19.

[96] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 19–35, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-32055-5.

[97] Peter Weidenbach and Johannes vom Dorp. Home Router Security Report 2020. Technical report, Fraunhofer FKIE, June 2020. URL https://www.fkie.fraunhofer.de/content/dam/fkie/de/documents/HomeRouter/HomeRouterSecurity_2020_Bericht.pdf.

[98] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 363–376. ACM, 2017. doi: 10.1145/3133956.3134018. URL https://doi.org/10.1145/3133956.3134018.

[99] Kazu Yamamoto. pgpdump: a PGP packet visualizer. URL https://github.com/kazu-yamamoto/pgpdump.

[100] Jonas Zaddach, Luca Bruno, Aurélien Francillon, and Davide Balzarotti. AVATAR: A framework to support dynamic security analysis of embedded systems' firmwares. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014. URL https://www.ndss-symposium.org/ndss2014/avatar-framework-support-dynamic-security-analysis-embedded-systems-firmwares.

[101] Zero Day Initiative (ZDI). Zdi-20-709, zdi-can-9768, (0day) netgear r6700 httpd strtblupgrade integer overflow remote code execution vulnerability, 2020. URL https://www.zerodayinitiative.com/advisories/ZDI-20-709/.

[102] Li Zhang, Jiongyi Chen, Wenrui Diao, Shanqing Guo, Jian Weng, and Kehuan Zhang. Cryptorex: Large-scale analysis of cryptographic misuse in iot devices. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2019, Chaoyang District, Beijing, China, September 23-25, 2019*, pages 151–164. USENIX Association, 2019. URL https://www.usenix.org/conference/raid2019/presentation/zhang-li.

[103] Lipeng Zhu, Xiaotong Fu, Yao Yao, Yuqing Zhang, and He Wang. Fiot: Detecting the memory corruption in lightweight iot device firmware. In *18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 13th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2019, Rotorua, New Zealand, August 5-8, 2019*, pages 248–255. IEEE, 2019. doi: 10.1109/TrustCom/BigDataSE.2019.00041. URL https://doi.org/10.1109/TrustCom/BigDataSE.2019.00041.