NURSE

eNd-UseR IoT malware detection tool for Smart homEs

Antoine d'Estalenx

TU Delft

NURSE

eNd-UseR IoT malware detection tool for Smart homEs

by

Antoine d'Estalenx

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 24, 2021 at 14:00.

Student number:	5149371	
Project duration:	November 9, 2021 – August 24, 2021	
Supervisor:	Dr. ir. C. Hernandez Ganan	TU Delft
Thesis committee:	Dr. ir. S. E. Verwer,	TU Delft
	Dr. J. G. H. Cockx,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

IoT devices keep entering our homes with the promise of delivering more services and enhancing user experience; however, these new devices also carry along an alarming number of vulnerabilities and security issues. In most cases, the users of these devices are completely unaware of the security risks that connecting these devices entail. Current tools do not provide users with essential security information such as whether a device is infected with malware. Traditional techniques to detect malware infections were not meant to be used by the end-user and current malware removal tools and security software cannot handle the heterogeneity of IoT devices. In this report, we design, develop and evaluate a tool, called NURSE, to fill this information gap, i.e., enabling end-users to detect IoT-malware infections in their home networks. NURSE follows a modular approach to analyze IoT traffic as captured by means of an ARP spoofing technique which does not require any network modification or specific hardware. Thus, NURSE provides zero-configuration IoT traffic analysis within everybody's reach. After testing NURSE in 83 different IoT network scenarios with a wide variety of IoT device types, results show that NURSE identifies malware-infected IoT devices with high-accuracy (86.7%) using device network behaviour and contacted destinations.

Preface

Conducting a master's thesis already is a complex project in itself, doing it during the COVID-19 crisis was one of the most challenging tasks I ever encountered. When I came to study in the Netherlands, I could not expect to spend a year and a half studying from home. However, I came here with basic knowledge about cyber security and I learnt a lot during these two years. In particular, my thesis project taught me a lot about botnet detection and IoT security.

I would like to thank Dr Gañán for supervising this project. Even though we both worked from our homes, his being always available for discussion and his helpful advice really helped me a lot during my project. I also appreciate a lot his help to better understand the research world to which I was unfamiliar, being from a engineering background.

I also thank all my friends for their constant support during this project, Dutch friends in Krashna and French friends through our online calls. Thank you for giving me this well needed balance between work and recreation.

Finally, I would like to thank my girlfriend for being there with me and supporting me during this though period. I also thank my family for their constant encouragements which gave me the strength to keep going until the end of this project.

Antoine d'Estalenx
Delft, July 2021

Contents

List of Figures

Acronyms

API Application Programming Interface.

ARP Address Resolution Protocol.

AS Autonomous System.

C&C Command and Control.

CDN Content Delivery Network.

CPU Central Processing Unit.

DDoS Distributed Denial of Service.

DGA Domain Generation Algorithm.

DHCP Dynamic Host Configuration Protocol.

DNS Domain Name System.

DoS Denial of Service.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IDS Intrusion Detection System.

IoT Internet of Things.

IP Internet Protocol.

IRC Internet Relay Chat.

ISP Internet Service Provider.

IXP Internet eXchange Point.

MAC Media Access Control.

MITM man-in-the-middle.

NAT Network Address Translation.

NURSE eNd-UseR IoT malware detection tool for Smart homEs.

QUIC Quick UDP Internet Connections.

RAM Random Access Memory.

TCP Transmission Control Protocol.

TLD Top Level Domain.

TLS Transport Layer Security.

TTL Time To Live.

UDP User Datagram Protocol.

VPN Virtual Private Network.

1 Introduction

1.1. Background

1.1.1. The Internet of Things

The Internet now gives the possibility to connect all sorts of devices across the world. This variety of devices leads to the emergence of a new expression: the *Internet of Things (IoT)*. While it can be used to refer to all devices over the Internet, it is more often used to describe sensors and devices that collect data and send it over the Internet. Kevin Ashton who introduced the expression in 1999 describes it as an environment where all the ”things” collect data about the physical world, thus linking the physical world humans interact with, with the Internet and computing power [11]. The collected data can then be used to better understand our physical world and how we interact with it. The IoT also made it possible to design ”smart” devices which use their sensors to adapt to our expectations as they ”collect, exchange and process data in order to adapt dynamically to a context” as described by the ENISA [22].

The Internet of Things is a growing part of the Internet and outgrows non-IoT devices over the Internet. In 2020, the number of connections from IoT devices surpassed the connections from non-IoT devices with about 54% of connected devices being IoT at the end of 2020 [45]. It is expected that this growth continues and that there will be almost 4 IoT devices per person in average by 2030.

1.1.2. Botnets

However, not everything about connecting devices to the Internet is beneficial. In fact, the additional communication capabilities create more entry points in the device, enlarging the attack surface and causing more security risks. The lack of security features in IoT devices could cause sensitive data to be stolen, or give the attacker the possibility to remotely control the device [61]. One of the possible misuses of insecure IoT devices is to use them in botnets. Botnets (from ”robot” and ”network”) are networks of compromised devices called ”bots” to which an attacker can send instructions remotely [38]. These botnets can be used to perform malicious activities such as sending spam, click fraud, hosting of illegal content, or performing Distributed Denial of Service (DDoS) attacks.

DDoS attacks are attempts to cause a service to be unavailable for some time by flooding it with internet traffic. A famous example of a botnet which performed DDoS attacks is Mirai. This botnet consisted of devices which had been infected by a malware using default username and password values [40]. Its first observed attack was against OVH, a French hosting provider, and reached a volume of 1Tbps [43], involving about 150 000 IoT devices. Since then, even bigger DDoS attacks have been observed, sometimes exceeding 2Tbps and targeting major services such as Google, Amazon Web Services or governmental institutions in Belgium [49, 64, 54]. Botnet DDoS attacks are getting more and more frequent and the grow of IoT means that more devices may possibly be infected and converted to bots, leading to bigger armies.

1.1.3. Botnet mitigation

The first step to prevent the rise of botnets and the multiplication of DDoS attacks could be to patch the vulnerabilities they use to infect devices. However, already infected devices would stay infected until

they are cleaned (for example by reinstalling the firmware). Therefore, detecting devices that are part of botnets and cleaning them are two required steps to eradicate a botnet.

Internet Service Providers (ISPs) are trying to mitigate the botnet spread among their network. There exist many botnet detection techniques to detect malicious activities and flag infected devices in ISPs network with very high accuracy. When an IP address is detected as malicious, the ISP is notified and can take action. The two most common measures from ISPs are to send a notification inviting the owner of the device to clean it, or to place the host in a walled garden to isolate it from the rest of the internet and prevent further harm from this host.

1.2. Research objectives

1.2.1. Problem statement

When notifying customers that malicious behavior has been observed from their devices, ISPs noted that users may not be aware of all the devices in their home that are connected to the internet [5]. Although some have antivirus software to detect malware for their computers, they may not realize that their IoT devices may also be infected. Therefore, asking them to identify and clean the infected device may be unreasonable expectations. While antivirus softwares propose easy to use solutions to detect infections on computers, there is no such solution for infected IoT devices.

Developing antivirus software for IoT devices is a challenging task [76]. The heterogeneous aspect of the Internet of Things would make it hard to develop a software that runs on all devices. Also, the limited resources and computer power would set constraints for such a software. However, many researchers have designed solutions to detect malicious traffic with high precision, which could detect the infection looking at the traffic from a device. Even if these solutions exist, they are not used in homes to help users with their infected devices and users still own devices that are part of botnets unbeknownst to them. As far as we know, no botnet detection technique has been designed with the use in home networks as an objective, and users have no available easy to use solution to detect the infected devices they own. In this research we will study the current botnet detection techniques, understand their requirements and attempt to develop a botnet detection technique for home users.

1.2.2. Problem relevance

Academic

There exist many botnet detection techniques in the literature (see section 2.1). Researchers also designed tools to detect malicious activities in IoT devices in the context of smart homes [10, 19]. However, we note that these solutions often rely on dedicated hardware, for example a modified router or a Raspberry Pi that is setup as a network middle box or has traffic mirrored to it and runs the software to analyze the received traffic. All these studies also assume that the user will know how to setup the software and network to ensure that the traffic is correctly rerouted. Yet, studies have shown that most users have limited internet skills [32] which makes this assumption unreasonable.

Our contribution in this report are the following:

- We design and develop NURSE (eNd-UseR IoT malware detection tool for Smart homEs): a tool to detect malware-infected IoT devices within home networks. Our solution does not require neither modifying nor adding new hardware components to the network, becoming the first malware detection system for IoT home networks.
- We identify characteristic features and behaviors of malware infected hosts. We then integrate these features in the detection logic of NURSE to identify malicious activities conducted by IoT devices infected with malware such as denial of service, port scanning and brute forcing.
- We evaluate our solution using a dataset of 83 scenarios containing a different types of malicious and benign traffic with various IoT devices. These include a wide variety of IoT device types ranging from air purifiers and weather stations, to alarm systems and android TV boxes. The evaluation reveals that our solution classifies benign and infected devices with a 86.7% accuracy.
- We make the our tool freely available to all users to download and run in their network to analyze their IoT devices. We also make the source code available in open source in a public repository (link) for further extensions and testing by the community.

We finally sent our contribution as a paper (see appendix C) to the Joint Workshop on CPS&IoT Security and Privacy (CPSIoTSec), for it to be reviewed by other researchers.

Societal

Our project aims at making botnet detection techniques available to end users. Therefore, the first societal impact would be to provide a software that allows users to check whether the IoT devices they own are part of botnets. Users are used to antivirus softwares to detect malware on their computers, however, there are no antiviruses for IoT devices. Botnet detection techniques exist but they are mostly designed for ISPs or company networks. Installing these solutions in a home network is a complex task, and there are no available botnet detection solutions designed for home users. Therefore, our idea is to provide a botnet detection software that users can run on their home computer, to detect if the IoT devices they own are infected with botnet malware.

This project is also part of the MINIONS (Mitigating IOT-based DDoS attacks via the DNS) research project which aims at reducing the impact of DDoS attacks in collaboration with ISPs. Our research could help ISPs for their task of botnet cleanup, especially with the notification approach.

When ISPs detect some malicious traffic coming from a home network, they know that one device in the home network is infected. However, they do not know what devices are in the home network as all the devices are behind the same public IP address. The ISP also cannot inspect the traffic to identify the infected device more precisely without the user's consent. Therefore, the ISP lacks information about the devices, and the user lacks information about the infection. As the ISP cannot clean the IoT devices in the user's network, they can only notify about the infection and push the responsibility for cleaning the infected device on the user.

Since users have trouble in identifying the devices that could be infected, NURSE aims at reducing the information asymmetry between the ISP and the user.

Users could use NURSE preemptively: running it once in a while to check if their network is safe. Thanks to this, they could detect infections in their home, and clearly identify the infected device without having to be notified by their ISP.

The information asymmetry could also be reduced in the other direction. ISPs could also ask users with infected networks to run NURSE to provide more information about the devices they own and to identify the infected device. With such additional information, the ISP could provide more precise assistance on how to clean the device and how to keep the network safe.

1.2.3. Research questions

We summarize this problem into the following main research question: **To what extent can IoT devices that are part of botnets be detected in home networks?**. To address this problem, we introduce the following research questions.

RQ1 How to detect IoT devices that are part of botnets?: To develop a botnet detection software for home networks, we first have to study botnet detection. We will try to understand what their requirements are and if they can or cannot be used in home networks.

- What are common botnet detection techniques?
- What do they need to be applied?
- Why are they not used or usable in home network?

RQ2 Can botnet detection be performed in a home network?: Once we have studied botnet detection techniques and understood their limitations, we will try to design and implement a botnet detection technique for smart home networks. We divide this development into the following sub-questions:

- What are the requirements for a home based botnet detector?
- What can be captured and used as input for traffic monitoring in a home network?
- How can this be used for detection? Can this be used to apply botnet detection techniques?
- How effective is the botnet detection technique we developed?

RQ3 Can it work in any home? Our goal is to develop a software for home usage. However, home networks and configurations may vary a lot. We must therefore measure the compatibility of our solution.

- What are supported operating systems? Considering that Windows is the most common operating system our tool must work on this operating system.
- Can the installation be made simple, requiring no setup?
- How much permissions are needed?
- How much does our tool execution affect the device it runs on?
- How much does our tool execution affect the scanned devices?

1.3. Thesis outline

The structure of this report is the following:

- chapter 2 will describe the state of the art in botnet detection techniques. We will then study IoT device identification techniques to look at some analysis techniques that may be useful to study IoT traffic.
- chapter 3 will analyze some requirements for botnet detection. We will then list requirements that will be necessary for our application to be used in home networks
- chapter 4 will introduce the traffic interception technique we will use, and which data can be extracted using this technique
- In chapters 5 to 9, we will detail the way we implement NURSE. Chapter 5 will detail the general architecture. More details about the domain classifier and the alert raising modules will be given in chapters 6 and 7. Chapter 8 will present some ways to list domains to take advantage of the whitelist functionality. Finally, 9 will study the user interface, and the software installation.
- Chapter 10 will present the evaluation of our solution, first on the botnet detection task, and then on some usability aspects.
- Finally, chapter 11 will present the conclusions of our research project, reflections about its limitations and possible extensions for further work.

2

State of the art

In this chapter, we study the state-of-the-art techniques for two problems: botnet detection and IoT identification. We then introduce *IoT-inspector* which is designed for IoT identification and shows how researchers adapted the problem to move it to smart homes.

We will study the botnet detection techniques in order to better understand how botnet detection is performed, and what the detection techniques require to be applicable.

We will also study the problem of IoT detection, both to understand how our tool could help ISPs in understanding their users behaviors and the IoT devices they use. This problem is not directly linked with botnet detection, however, this research will be useful to better understand IoT traffic, especially normal IoT traffic which could be useful for whitelists. Finally, the problem of IoT identification lead researchers to develop IoT-inspector, a tool that allows users to examine the traffic of their IoT devices. While this examination feature was a side feature in their software, their project shows that traffic interception at home is possible and gives us an example on how it could be done.

2.1. Botnet detection

Botnet detection is a problem that has been studied a lot as the threat of botnets rose. The goal is to be able to detect hosts that are infected with a malware by monitoring their network traffic. There exist a lot of different botnet detection techniques. In 2009, a survey of botnet detection techniques [23] identified 4 different approaches: signature-based detection, anomaly-based detection, mining based detection and DNS-based detection. Even if the census is from 2009, the approaches are still relevant nowadays. In this section, we will describe these categories and present a solution that correspond to each.

Signature-based detection

Signature based detection is now the most widespread technique for malicious behavior in a network. It is mostly used in Intrusion Detection Systems (IDSs) that monitor traffic to find patterns that correspond to specific activities that reveal that a device is under attack, or that a device is infected by a malware. An example of such an IDS is Snort [60] which is open-source. Snort works with a set of rules that are predefined and can be tweaked by the user to flag some suspicious traffic. However, a limitation of signature-based detection is that it can only detect known attacks and cannot identify new threats since their signatures are unknown.

Anomaly-based detection

A second approach for botnet detection is anomaly-based detection. This statistical approach tries to identify abnormal events that could be caused by an attack. For example, a device that is part of a botnet could take part in a Denial of Service attack and suddenly send a lot of packets to a single host. Such a behavior could be quite abnormal for the device and an anomaly-based detection method could raise an alert to suggest that the device is acting oddly, suggesting that it may be infected. An extension of this detection method could be to use not only the temporal correlation, which helps detecting abnormal behavior at a certain time, but also spatial correlation, to note if many devices perform the same actions

at the same time. This approach is presented in BotSniffer [29] in which researchers aim at detecting discussions with command and control servers using spatial-temporal correlation. BotSniffer monitors the traffic, mostly keeping track of suspicious activities and use of protocols that are sometimes used to communicate with C&C servers (HTTP and IRC at the time of writing). It then logs all these activities and analyzes it with a correlation engine. Using statistic algorithms, they detect when multiple devices had similar activities that were logged, which could indicate that they are both part of a botnet. This method takes advantage of the fact that botnet malwares infect a lot of devices which then act similarly as the malicious code they run is the same. It therefore detects more C&C communications than signature-based systems which may miss some traces they do not know. It also produces very few false positives.

BotSniffer also deploys some techniques to detect single infected devices in a network, by comparing the behavior of clients in the IRC channels used, or by detecting periodical patterns for HTTP communications. However, researchers admit that these single-device targeted techniques are not as resilient as the ones involving multiple devices, since malware developers could add some variations to avoid detection. Anomaly-based detection are therefore better than signature-based detection in that they can detect new threats which signatures are unknown by noticing a new behavior in some hosts, however, they require more data, for example traffic from multiple devices to be able to detect abnormal behavior that could reveal combined queries.

Mining-based detection

The third approach for botnet detection is based on data-mining to find examples of malicious traffic and use machine learning to identify malicious traffic. In the survey, some examples of data-mining based detection techniques are presented, with techniques analyzing IRC communications to identify suspicious nicknames and servers or flag IRC flows that correspond to command and control communications looking at payloads. However, botnets are evolving and are now using other protocols such as HTTP, and sometimes even custom protocols. So data-mining solutions are now looking at networking flows, and one example of such technique is BClus. This method is presented in a comparison of botnet detection methods [26]. In this comparison, researchers present two of their botnet detection methods, one of which is BClus. This method analyzes the flows, and adopts a behavioral-based detection approach. The method works by analyzing flows, cutting them in time windows to reveal temporally local behavior, and then aggregating them by source IP address to reflect the behavior of each host. The flows are then aggregated in aggregation windows and researchers derive features out of these aggregated flows to perform clustering. As features were computed which allow to identify flows from infected and safe hosts, a classifier can be trained to recognize the flow clusters. Finally, researchers label some of the malicious flows to teach their classifier which of the clusters correspond to botnets, considering that clusters which malicious flow proportion exceed a threshold are labeled as botnets.

DNS-based detection

The final approach for botnet detection could be to inspect the traffic in a specific protocol to detect malicious behavior. This can be done with IRC or HTTP communications, however, such an inspection could easily be countered by encrypting commands before sending them. Therefore, one of the inspected protocols is Domain Name System. This protocol is useful to inspect for two main reasons: first because it is not encrypted and second because it is used a lot as it connects the domain names with IP addresses. Inspecting DNS gives the list of domains contacted by the device, and the IP addresses associated with it. This can be useful to detect abnormal behavior, such as contacting a lot of non-existent domains which can happen when a Domain Generation Algorithm (DGA) is used for to generate a meeting point domain. This can also be used to analyze the contacted domains, and check if some of these are malicious, for example by checking whether they were generated by DGAs, more details about existing DGA detection techniques will be given in section 6.1.2.

Another possible use of DNS is to build profiles of malwares via DNS. Dwyer et al. [21] designed a technique to detect domain names associated with Mirai hosts. DNS is used to analyze not only the domain itself in terms of entropy and vowel density, but also the associated response, with its Time To Live (TTL) and the IP addresses and associated Autonomous Systems (ASs). Some features were even derived from this response using external data such as geolocation of IP addresses to compute the spatial diversity of domains. Comparing different classifiers, the researchers noted that their random forest model did perform best with a 99% accuracy on average.

Datasets

An interesting contribution in botnet detection is the dataset introduced for a comparison of botnet detection techniques [26]. In this paper, researchers noted that while many researchers had designed botnet detection techniques, few of them compared their method with others. Also, many researchers had created their own dataset of malicious traffic, some including only malicious traffic while others also had background or normal traffic. Also, datasets were in different formats and were not all labeled. This lead the researchers to create the CTU-13 dataset which contains 13 scenarios representing different botnet activities. The goal is to provide a public labeled dataset that could be used by other researchers to evaluate their detection methods. They also used this dataset in the article to compare multiple botnet classifiers including BClus which they designed. Since the release of this dataset, more captures have been released to complete the dataset, and some more specific datasets have been created. This includes the IoT-23 [55] dataset which contains 23 captures of botnets on IoT devices performing malicious activities along with 9 captures of benign traffic. This dataset is the one we will use to evaluate the detection of our tool.

2.2. IoT device detection

2.2.1. Active scanning

Most of the botnet detection techniques described in the previous section do not depend on the host type, and could be applied indifferently to desktop computers or IoT devices. However, as the number of IoT devices is increasing rapidly and exceeds the number of non-IoT devices, it becomes an interesting target for malware, especially because IoT devices may have open ports such as telnet that can make them vulnerable to malware such as Mirai. In fact Mirai infected mostly IoT devices.

Therefore, botnet detection techniques could be adapted to better detect malware targeting IoT devices. However, applying these techniques would require to isolate IoT devices. This can be done when analyzing the traffic of a device in a controlled environment, but not at a higher level. That is why researchers are now working on detecting IoT devices among the internet. Some techniques even attempt to identify the manufacturer or the model, which could help in running more specific botnet detection tests. For example identifying a device which model is known to have an open telnet port could greatly help in detecting devices infected with malwares that spread through open telnet ports such as Mirai. IoT detection could also be coupled with botnet detection at an ISP level, to link the malicious traffic with a device model and send more precise cleaning procedures when notifying users.

A first approach to identify IoT devices among the internet could be to perform scanning, for example using Shodan [66]. This scanner allows to scan the internet for device with a specific port or banner and is often used by attackers to find devices that run vulnerable versions of softwares. The use of scanners to find devices of a certain type is illustrated in [47] where researchers show how Shodan, Masscan and Nmap can be used to find specific IoT models that are vulnerable. Similarly, researchers analyzed the network of CERN, trying to identify IoT devices using web-scraping [2]. They identified all devices and profiled them, first by scanning their open ports, and then by scraping their web interface when they had one. Using this technique, they managed to identify many IoT models and manufacturers. They then performed vulnerability assessment on the identified devices to note than 11% were vulnerable with default credentials, and than another 13% had easy to guess credentials or manufacturer's default credentials.

On a larger scale, a scan of vulnerable devices was performed in 2012 using the Carna botnet [37]. The botnet was not developed to be malicious, and simply ran a scanner on each device it infected. This botnet managed to identify and infect about 1.2 million devices using default credentials and used each infected host to find even more hosts to infect and to perform a massive port scan of the internet. While the botnet creator mostly wanted to analyze how big the internet was at this time (the botnet ran in 2012), it also provided a map of infected devices (see figure 2.1) which shows how vastly the botnet spread using a simple scan method.

The scanning approach is efficient at detecting devices, however, since most botnet detection techniques rely on traffic analysis, it may not be possible to perform a scan of the devices. The traffic may have been captured earlier and stored before analysis, or could be mirrored to the device analyzing the traffic, with no possibility to scan the device the traffic came from.

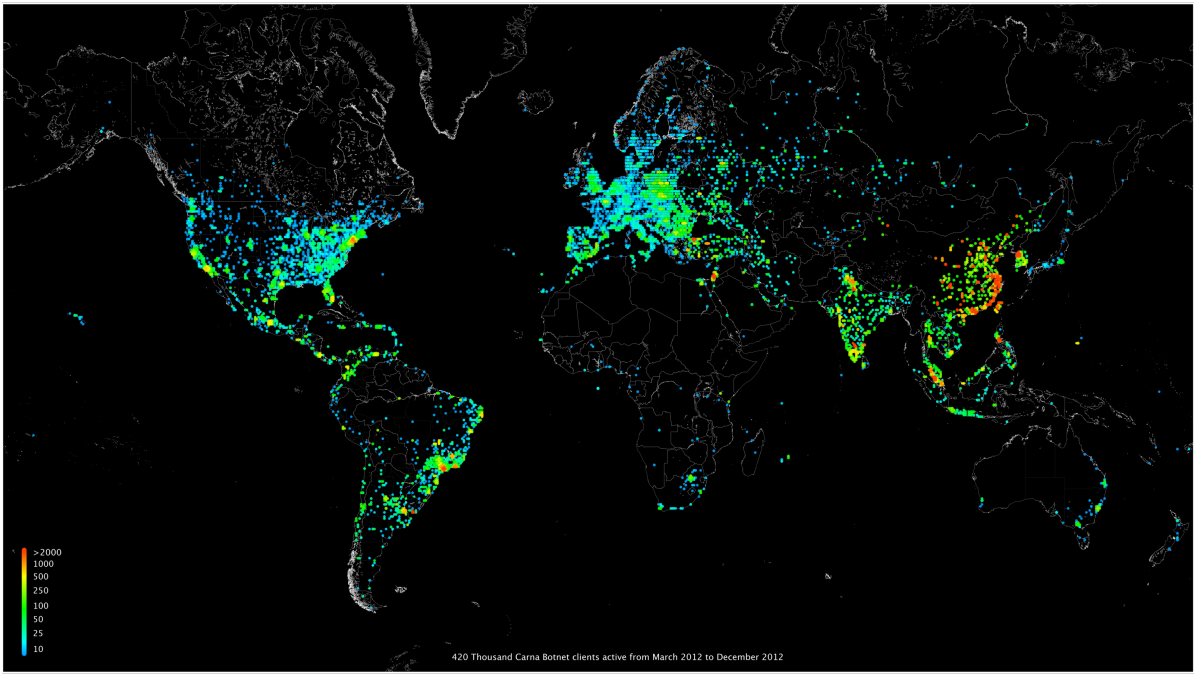


Figure 2.1: Locations of hosts infected by the Carna botnet

2.2.2. Detecting with traffic monitoring

A second interesting approach to IoT identification would be to identify IoT devices in traffic captures. Although it is a new research domain, some researchers have developed techniques to identify devices based on traffic captures. Most of these techniques are based on domain names that are contacted by devices since these can be easily obtained from captures, as they are not encrypted in DNS queries.

With IoTFinder [57], researchers from university of Georgia propose a model to fingerprint IoT devices behind a Network Address Translation (NAT) and identify them in an accurate and explainable way. Their idea is to profile each device with a list of domains associated with their query frequency. The researchers then compare the fingerprints of the device they want to identify with fingerprints of known devices to identify the unknown device. The method identifies a device based on a previously recorded profile quite accurately. The researchers also note that there is little confusion between IoT devices, except for some IoT models from the same manufacturer, for example Google Home and Google Home mini, which contact the same domains and therefore have identical DNS profiles. Their tool detects most IoT devices, only missing some when they were mixed with other traffic behind the NAT. Finally, the researchers used their tool on an ISP DNS traffic captures and performed identification among the 40 million ISP clients to provide the IoT device distributions.

Detecting IoT Devices in the Internet

In their article [30], Hang Guo and John Heidemann attempt to detect IoT devices to measure the growth of IoT devices. IoT devices sales are growing and the security vulnerabilities in these devices can lead to massive DDoS attacks. That is why the researchers want to detect IoT devices and measure their distribution over the internet. They propose 3 detection techniques: IP-based, DNS-based and TLS-based. The IP based technique works by listening to DNS traffic of 10 devices they bought to establish profiles of contacted IP addresses. They then set a threshold and assign a label to a unknown device is it share 2/3 of its contacted IP addresses with a known device. This technique works well on their devices, but researchers note that it is not time resilient as IP addresses assigned to domains can change over time.

The second technique is DNS-based. It is similar to the previous one, except that working with the domain names instead of IP addresses prevents the changes over time and allows to identify third-party domains and manufacturers domains thanks to domain name and whois information. The techniques shows great accuracy over their devices, and seem to be more time resilient since it detected devices in several years old traffic captures.

Finally the TLS based detection method attempts to identify IoT devices that provide an HTTPS interface (such as IP cameras). Researchers analyze the TLS certificates of the web page and search for keywords that could identify the manufacturer or device type. These 3 techniques show accurate detection in the experiments carried by the researchers, however, they are limited to detecting the device that were in the training set.

Detecting IoT Devices at an ISP level

The previous study mostly performed detection on traffic captures at local scales, for example on university campuses. Other researchers studied the detection of IoT devices at a higher level: moving the detection at the Internet Service Provider (ISP) or Internet eXchange Point (IXP) level [62].

The detection is separated in 3 levels:

- Platform-level: being able to identify a firmware or the backend infrastructure, which may not be specific to the manufacturer. For example, identifying devices which use AWS (Amazon Web Services) infrastructures [6].
- Manufacturer-level: identifying the device manufacturer
- Product-level: the finest detection level: identifying the product type. This detection level means being able to separate connected TVs made by a manufacturer from other devices from the same manufacturer.

The detection technique relies on detecting IoT-specific infrastructures. The researchers first identify domains contacted by devices by monitoring DNS traffic and classify them into IoT specific domains and generic domains. They then obtain the IP addresses associated with IoT specific domains using DNSDB [63], and filter out the ones that are shared between multiple services to obtain the dedicated infrastructure. Finally, the endpoints (IP address, port) are associated to the device which creates a profile for each device type. The detection rule to label an unknown device is based on a threshold set as a proportion of the number of associated domains for each device type, for example labeling as product A if more than 40% of the endpoints observed in training with product A have been contacted.

Tested using crosscheck validation, the detection reached 72/93/96% of detection in 1/24/72h, and about 75% detection rates for devices in in idle mode, which shows that the detection method needs a time window of some hours to obtain enough traffic and return an accurate classification. When run at the ISP level, the tool detected many Amazon and Samsung based devices. Running at the IXP level, the researchers could note daily activity patterns as the number of active Alexa based devices followed diurnal patterns, showing when they are used by humans. However, other devices, for example TP-link plugs, generate a low traffic volume, which makes them hard to detect.

This method achieves great detection performances when evaluated, however, researchers acknowledge that it is limited as it cannot identify devices which were not in the training set, and does not work well for devices that have limited network traffic.

2.3. Moving detection tools to home

The previous section presented how researchers attempt to narrow botnet detection by detecting IoT devices in the internet. However, this is not an easy task, and most of the current techniques are limited as they only detect devices they have been trained to detect.

Since researchers cannot own all devices, researchers from Princeton university have proposed the idea to crowdsource IoT identification in *IoT-Inspector* [35]. In this article, they proposed *IoT-Inspector*, a tool aimed at collecting device traffic to create a dataset for IoT identification. The tool is meant to be run by volunteering users who own devices on their own computer. This addresses two issues with IoT detection datasets: scaling, as the tool can be downloaded by anyone, thus giving a potential access to any IoT device, in real conditions; and labeling as the users are asked to provide the device model when adding a device in the application.

IoT-inspector exploits ARP spoofing to intercept traffic, collects data about the traffic (such as MAC address, contacted IP addresses, bytes sent and received...), anonymizes it and sends it to the university servers, thus constituting a dataset of IoT traffic. In January 2020, about 10,559 users had downloaded the tool and registered 182,630 devices, out of these, 5,404 users had allowed traffic monitoring, creating a dataset of traffic captures for 54,094 devices.

To create a dataset as comprehensive as possible, the researchers have to reach as much users as possible and convince them to allow traffic monitoring on their devices. That is why they designed a tool that also serves the user's interests to help users in detecting privacy breaches. The tool analyzes the traffic and displays the contacted domains with some additional information such as the domain owner or whether the domain is known to host trackers. It also presents charts plotting the amount of data sent to each of these domains by the device. With these graphs, the researchers aim at giving new insights to the users and help them in detecting privacy breaches by visualizing where their data is sent.

IoT-inspector thus moves the IoT-detection right in the home of the user. This has many advantages:

- the IoT traffic can easily be isolated
- the user can run the inspection by himself, with no need of additional device or more computer power.

This tool also moves the privacy breach detection at home, allowing users to have a look at their own device to detect privacy breaches. With the provided insights, users could for example detect an infected IoT device in their home by noting that some data is sent to domains associated with malwares.

2.4. Conclusion

There exist many botnet detection techniques which have proved to be efficient, and as most rely on traffic analysis, some researchers proposed datasets of malicious traffic to compare them. The techniques rely on traffic observation and on knowledge of previous attack patterns or anomaly detection. Therefore, they often need to run in controlled environments, where traffic can be forwarded or mirrored. They also may not be easy to set up when additional data for attack patterns detection (for example Snort rules) has to be downloaded.

Most of these techniques are targeting malwares in general, but, as IoT devices become a growing part of the internet, some researchers work on identifying devices to help in measuring this growth and in targeting these devices for more specific IoT botnet detection. This could help in detecting infected devices in homes from the outside, by being able to link the device identification with the botnet detection technique. However, the IoT identification techniques are not yet comprehensive enough to be able to identify any device, as they often rely on training data from a device to identify its model in the wild.

IoT-inspector was designed to collect data for IoT identification from the users' home networks. However, it also provided some traffic inspection functionalities to allow users to detect privacy breaches from devices in their home. While these functions were added as incentives for users to use their data-collection software, this shows that traffic inspection techniques can be bundled into softwares to be run in home networks.

3

Requirements for botnet detection at home

We presented multiple botnet detection techniques in chapter 2. In this chapter, we try to understand their limitations, and why they are not suitable for home usage. We will then elaborate a list of requirements for our solution to be used in home networks.

3.1. Requirements for botnet detection

3.1.1. Type of data and features used

Every botnet solution that we listed in chapter 2 starts from the same data input: the network traffic. The way it is processed depends on the approach. In Intrusion Detection Systems, rules are applied to detect suspicious flows among the traffic. In other approaches, more advanced analysis is conducted, for example in Botsniffer which analyzes the correlation between hosts traffic.

Some of the methods we listed analyzed the packets contents, to recognize some signatures of malwares or some suspicious behaviors, for example in IRC channels. However, nowadays, most of packet contents are encrypted. Even if DNS is not encrypted, which makes it possible to use DNS-based approaches to detect suspicious domains, HTTP has been replaced with HTTPS [28] which encrypts the packet content, making packet content analysis impossible. That is why, some analyses are limited to packet headers which are not encrypted.

Traffic data may not always be enough for botnet detection, and some solutions may rely on external data. For example, many of the DNS-based solutions use external data to obtain more information about the contacted domains. In Domain Generation Algorithms detection, an example of such additional data is passive DNS which is a database of all DNS records seen over time. Such data can be very helpful to analyze the evolution of a domain over time, and therefore feed the botnet detection method with features that cover multiple years without having to run the analysis for this duration. We also noted that in some botnet detection methods, additional data could be used to provide additional features such as geographical features as seen in *Profiling IoT-based Botnet Traffic using DNS* [21].

3.1.2. Data acquisition

We listed the inputs and required data for existing botnet detection techniques. In order to understand whether these techniques could be adapted for home networks, we have to study how this input data can be obtained.

All the botnet detection techniques that rely on traffic data consider that the traffic of the analyzed machine can be intercepted. This is understandable if the botnet detection technique is designed to be applied in an ISP network or in a company. In these networks, the administrator has complete control over the network and can easily reroute the flow or mirror it to the host that runs the botnet detection method. The botnet detection methods we listed also have advantages that make them quite suitable for ISPs and companies. For example, BotSniffer correlates the behavior of all hosts to detect similarities via correlations, which requires a large number of hosts which is why they ”recommend a distributed deployment of BotSniffer to cover a larger network space” [29].

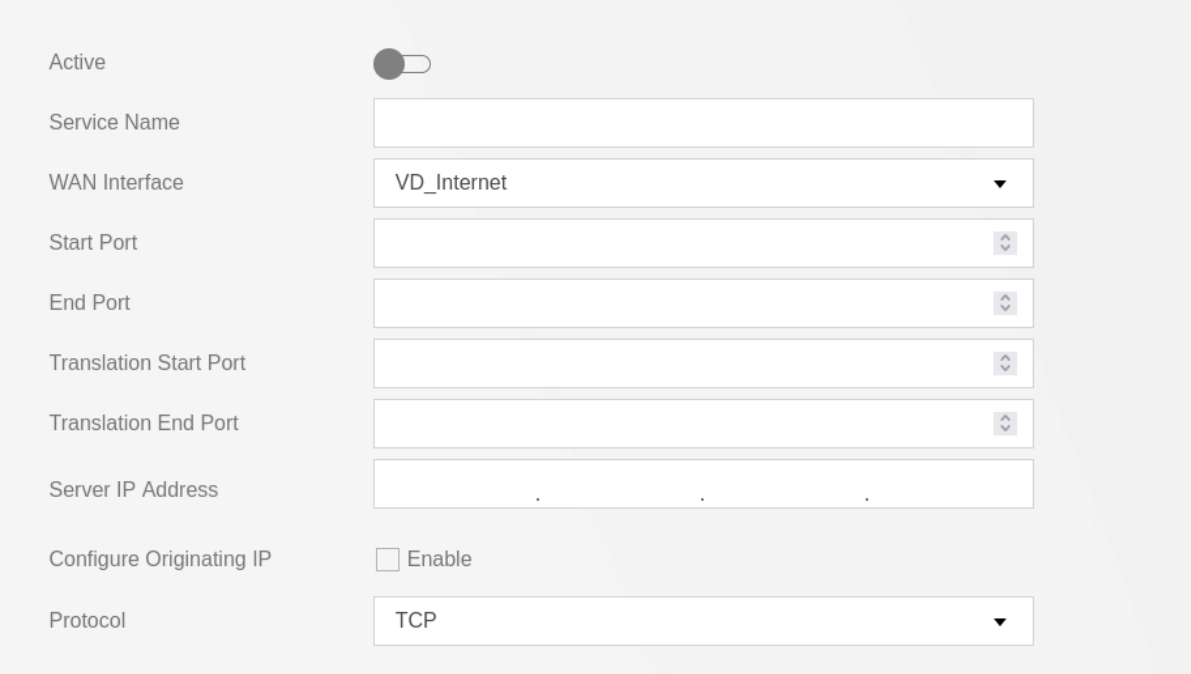


Figure 3.1: The port forwarding interface of a router

Traffic interception may however not be easy to setup in a home network. Some routers offer the possibility to set rules to forward traffic as shown in figure 3.1, but this functionality may not be available on all routers. But the main limitation in home networks does not lie in the router functionalities. As some users may not even know which devices are connected to the internet in their homes (see section 1.2.1), we assume that they would not know how to setup traffic interception or mirroring for a specific device. Even with a detailed explanation of the fields seen on the traffic mirroring window shown in figure 3.1, the users would still have to know the IP addresses of the devices they want to monitor.

A possible workaround to analyses requiring live packet forwarding could be to perform the data acquisition separately from the botnet detection, save it and analyze it after the capture. However, this still require to be able to intercept the traffic. Some softwares such as Wireshark [75] offer this feature, but can only intercept the traffic of the host they run on. This software cannot be installed on IoT devices such as a smart lamp, but could be used on the router which would provide a capture of all the home network traffic which could be used to perform botnet analysis. This approach is however complex to set up, and may even only be doable via command line (for example via telnet on some Zyxel routers [72]). Another limitation of this approach is that the analyzing tool would not have access to information that can only be obtained in live captures, such as device names. Although this information may not be necessary for the botnet analysis (as we only listed traffic data and some external databases as requirements for botnet detection), it could be helpful for users who cannot identify the devices in their home from their IP addresses.

Another limitation of botnet detection techniques in terms of data acquisition lies in the use of external data. The external data that botnet detection methods use is easier to compute on a large scale network. Passive DNS databases for example are often computed by ISPs which have networks of millions of users which allows to gather a lot of DNS records. Large companies could also compute their own passive DNS data instead of relying on external databases, but would maybe suffer from a bias since all of their users work for the company and therefore may have similar browsing patterns. However, home users cannot generate large passive DNS databases. This databases require time to be gathered, a large number of users to be diverse enough, and use a lot of memory since all DNS records have to be stored.

For those who cannot gather passive DNS databases or access other external databases such as IP geolocations, some are available via online APIs such as DNSDB [63]. However, these APIs are not always freely available and may be licensed. These commercial databases could be a solution for a company which wants to invest in its botnet detection method. However, home users may not want to

buy licenses for such databases. It is also not possible to bundle this databases into a botnet detection software for home usage, both because of the licenses on commercial databases and because the databases may be several terabytes big.

3.2. Requirements for home usage

Now that we have identified the limitations that cause botnet detection to be hard to use in home networks for users, we will try to list requirements for our solution since we want to design it for smart homes usage.

As we mentioned in the previous section, running the analysis as the traffic is captured and in the capture environment could allow our botnet detection solution to obtain additional information about the devices such as the device names via local DNS. Following this approach, we could think about performing more advanced scans of the devices, for example port scans. However, active scanning could also cause some issues in the network. For example, the legal issues of Nmap mention that port scanning may cause crashes on some systems. As we do not know how stable the IoT devices in the network will be, we will not attempt any active reconnaissance that could cause some devices to crash. Therefore, the device detection in NURSE will be passive. Instead of pinging all addresses in the IP range, we will listen to traffic to identify IP addresses that are active. This approach may be limited as it will not identify devices that are idle, however, we consider that if a device does not send any traffic, it is not necessary to scan its behavior to check whether it is part of a botnet or not.

Other than having to be passive, our botnet detection technique will have to be designed with some limitations associated with the use in home network in mind. The limitations will come from multiple aspects:

- NURSE has to be used by non computer literate users
- NURSE has to be installed on a user's computer in the network. The computer also cannot be considered to be dedicated to this task and has to remain usable while the tool is running.
- NURSE has to run in a home network, and require no networking setup.

From these conditions, we derive the following requirements for our software:

- *No router configuration*: we cannot require a specific router configuration for our botnet detection method to work. This limitation comes from the fact that users may not know how to configure their router accordingly. The diversity of router models also makes it hard to provide detailed instructions for users on how to set up their router.
- *Limited computer setup*: NURSE will obviously have to be installed on the computer, but we will try to make the installation simple. We also do not want any operating system settings to be required for our software to work. This is both because the settings may be too complex for users, and because changing settings system wide may affect the usability of other application on the same computer.
- *Limited space, CPU and bandwith usage*: we cannot assume that users have a dedicated machine for botnet detection, and it is very likely that they will run NURSE on a computer that they use for other applications. Therefore, we will try to limit the disk usage, the CPU usage and the network bandwith affect so that the computer remains usable when our software is running.
- *Platform support*: we will try to develop a tool that is compatible with most operating systems, especially with Windows as it held about 87,56% market share in the end of 2020 [51].
- *No network disturbance*: NURSE should not affect the network functionality as it runs. This is important both to ensure that devices remain functional, and also to limit evasion, as malwares could try to stop their malicious activities if they can detect that NURSE is monitoring their activity.

IoT-inspector shows how researchers from Princeton university attempted to make traffic inspection available to home users. Their tool has recently been made available for Windows and can be run from source code with a python environment. It respects most of the conditions we listed, but we noted two limitations. First, the data is sent to the servers online, which takes some network bandwith and

may slow the connection in case the user has a low bandwith. We also noted that they used ARP spoofing as a capture method, which matches most of our requirements, but relied on IP forwarding at the operating system level, and therefore modified the operating system network settings. This method works well for traffic interception and in the next chapter we will detail how we built our traffic capture method inspired from this one but for botnet detection.

4

Capturing traffic in a home network

4.1. Goal of the capture method

4.1.1. Why traffic capture

As seen in the state of the art (chapter 2), most botnet detection methods rely on traffic inspection. They therefore require to have traffic redirected or mirrored to the device that performs botnet detection.

However, we aim to design a tool that can be used in a home network. While it may be possible to set up a traffic mirror in some routers, users may not know how to set this up. It is also likely that users who know how to setup their network with traffic mirroring can install one of the botnet detection methods we mentioned earlier. To make botnet detection easy to set up for home users, we therefore try to cover this networking setup and to achieve the traffic inspection with no additional setup required.

As changing the router networking setup from a host is not possible for security reasons, we instead try to perform the redirection ourselves. This also makes us less dependent on the house configuration as interactions with the router could depend on the router model, which would cause issues if not all router models are supported.

In this chapter we introduce the traffic capture technique we will use to intercept traffic from the monitored devices.

4.1.2. Technical requirements

There exist multiple methods to intercept the traffic of a device in a network [39]. These are often used by an attacker to set himself in a man-in-the-middle (MITM) position, in which he can intercept and sometimes even modify the packets as they are sent.

One possibility could be to set up a proxy on the device, for example to perform HTTP interception with Burp Suite [59]. This allows the attacker to intercept the HTTP traffic and even to modify the packets before forwarding them. This method makes traffic interception easy to setup and allows attackers to set up a MITM over a browser or an application, but is limited: it first requires to have access to the device networking settings to set up the proxy; and is limited by the software, as Burp is mostly designed to intercept HTTP traffic. We cannot ask users of NURSE to setup a proxy on their device: this requires some technical knowledge the user may not have, and may not even be possible since manufacturers may not give access to the network settings of their device, for security reasons.

Therefore, our capture method needs to:

- intercept all packets of all protocols and not to be restricted to a specific protocol
- require no setup on the targeted device, as the user may not know how to setup the interception, or may not even be allowed to.
- require as little configuration as possible on the user's machine
- to not disturb the user machine (allow the user to browse the internet while the tool intercepts traffic)

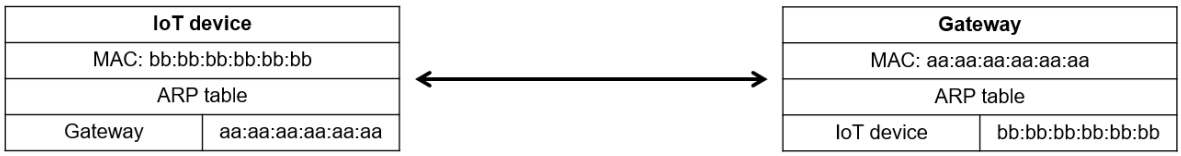


Figure 4.1: ARP table of two devices in a local network

- to be transparent and not disturb the IoT traffic: if intercepting the traffic cause networking problems, the intercepted traffic will not reflect the device normal behavior. Also malwares could detect that they are being monitored and stop malicious activities

4.2. ARP spoofing

4.2.1. ARP protocol

The Address Resolution Protocol (ARP) protocol is a protocol defined in RFC 826 [17] to map the internet layer, using Internet Protocol (IP) addresses to the link layer which uses different addresses such as Media Access Control (MAC) addresses.

When a device sends packets to a device with a given IP address in a local network, it has to specify the MAC address of the destination. However, they may not know this MAC address. Therefore, the sender broadcasts an ARP request to all devices (using the broadcasting MAC address `ff:ff:ff:ff:ff:ff`), asking who has the destination IP address. The device with the looked up IP address replies to this request with an ARP reply which contains its IP and MAC addresses. Thanks to this, the first device learns the MAC address associated with the IP address and can send its packet.

The ARP protocol also allows devices to advertise the IP they have by broadcasting their IP and MAC addresses, so that every user can note the association. This gratuitous ARP message is also particularly useful as most devices have an ARP table which caches the associations, so that devices are not required to send an ARP request before each packet they send.

Figure 4.1 presents the normal state of ARP tables for two devices in a local network. Each device stores the IP and MAC address of the devices it communicates with.

The Address Resolution Protocol however does not provide any method to authenticate a message, which means that any device in the network can advertise any address association, leading to some exploits such as ARP spoofing.

4.2.2. ARP spoofing

ARP spoofing is a technique used by attackers that exploits the fact that ARP is not authenticated. Using it, attackers can pretend to have another address than their own, and can intercept some traffic. By combining multiple spoofs, an attacker could place himself in a man-in-the-middle position.

To perform ARP spoofing and intercept traffic sent to a victim device, the attacker only has to send an ARP packet advertising an association between his own MAC address and the IP address of the victim. This can be done either by sending gratuitous ARP packets to advertise the associations, or by answering to ARP requests before the victim so that only the attacker's reply is considered (only the first reply to an ARP request is taken into account).

By performing a double ARP spoofing, impersonating both the router and a device in the network, an attacker can receive all the packets from the device to the router and from the router to the device. By simply forwarding the packets to their intended recipient, he can set himself in a man-in-the-middle position, being able to intercept all the communications of the victim with the router and therefore with the internet.

Figure 4.2 shows how an attacker can perform ARP spoofing to get into a man-in-the-middle position to intercept the communication between the devices in figure 4.1. The red fields in the ARP tables show the values that have been spoofed by the attacker in order to intercept traffic. We note that this attack requires the attacker to know the IP and MAC addresses of both devices.

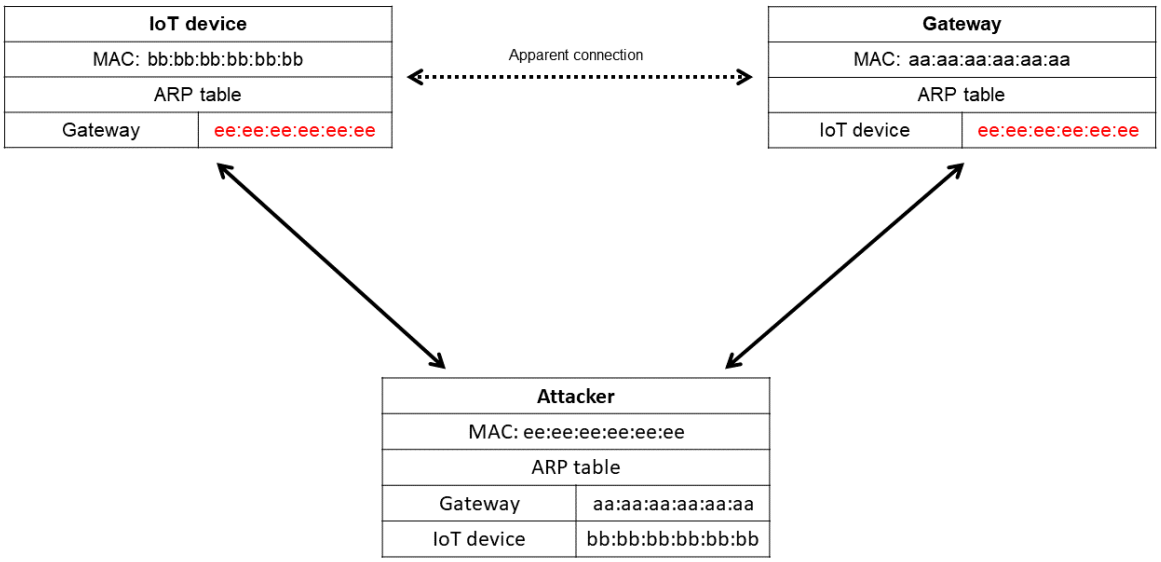


Figure 4.2: ARP spoofing in a local network

Figure 4.3: Wireshark alert message for duplicate use of IP address

4.2.3. Advantages and limitations

ARP spoofing is an easy to set up attack, that allows an attacker to fully intercept the internet traffic of their victim when they are in the same network. Therefore, it suits our needs for traffic capture as it allows us to target a device and intercept its traffic without having control over the targeted device. This attack is also easy to implement as it may not require many lines of code when using libraries to send ARP packets. For example, using the python library Scapy, an ARP spoofing can be performed in less than 100 lines of Python code [44].

The ARP spoofing technique also does not require any control over the victim device, therefore, it works with any device in the same network as the attacker, without needing a set up on the victim device which may not be possible as we are targeting IoT devices.

It is also transparent, and is not immediately noticeable by the victim. The only way to detect ARP spoofing on the victim side is to check the ARP table to notice that an IP address has been claimed by two different MAC addresses. Some more advanced detection tools can keep track of ARP associations and display alerts in this situation. For example, the warning displayed by Wireshark [75] (a network sniffing and analyzing tool) is shown in figure 4.3. As the attack also adds a hop on the network, it may slow the network speed which could make it noticeable for some users.

ARP spoofing impact is fortunately limited because it cannot decrypt traffic, meaning that only non-encrypted packets can be read by the attacker. As HTTPS is more and more adopted [24], reaching over 95% of traffic for Chrome users in 2020 [28], most of the HTTP traffic is now encrypted, which reduces the impact of ARP spoofing. Non-encrypted protocols such as DNS and packet headers can still be intercepted which will reveal the activities of the targeted devices without breaching users' privacy as their personal information remains encrypted.

4.2.4. IP forwarding

IP forwarding is a functionality of operating systems that allows the system to transfer IP packets. When enabled, if the operating system receives a packet which destination IP is not its IP address, it forwards it to the actual recipient (following its own ARP table). This functionality is very useful to set up the man-in-the-middle position with ARP spoofing and is often present in ARP spoofing implementations [44,

35] since the attacker keeps receiving packets that are not destined to its device. With IP forwarding enabled, the operating system takes care of forwarding these packets instead of dropping them, which ensures that the packets arrive to their destination, and that the attacker does not blocks the connection he wants to intercept.

As this forwarding is performed at the operating system level, it is quite fast, however, this also means that the attacker can only sniff the packets as they pass through, thus ending in a man-on-the-side position. While a man-on-the-side position could be enough to monitor traffic, we wanted to add some possibilities for the user to take action if an infected device is detected in his network, for example by blocking malicious traffic. To achieve this, we used our own implementation of IP forwarding at the application level, using Scapy [14]. This allows us to open every non-encrypted packet we intercept with Scapy and to modify its contents before forwarding it. This also makes our ARP spoofing less intrusive and operating-system dependent as enabling IP forwarding requires to modify some network parameters on the user machine.

A drawback of this re-implementation is that packet forwarding is slower than when performed at the operating-system level. However, we considered that the added benefits of being able to tinker with the intercepted traffic could outweigh the network speed loss. This re-implementation also makes the ARP spoofing script more portable since enabling IP forwarding is a procedure that differs depending on the operating system, and may require additional permissions.

4.3. DNS spoofing

In the previous sections, we described ARP spoofing, which will be the technique we will use to intercept traffic for botnet detection. But before, we present the botnet detection part using the traffic interception, we illustrate the domain blocking functionality of NURSE. This functionality does not aim at detecting infected devices, but can help the user in taking action once a device is detected as infected by our tool. This functionality is also an illustration of how we use our custom IP forwarding (see section 4.2.4) to modify packets as we intercept them.

As explained in section 4.2.3, ARP spoofing allows us to read the contents of non-encrypted packets. Among the non-encrypted protocols is DNS, which means that we can easily intercept, parse and modify the DNS packets before forwarding them. This allows us to perform DNS spoofing i.e. to alter the DNS responses to change the IP addresses associated with a domain. This attack is powerful, as it means that we could redirect traffic going to a specific domain to any IP address of our choice. As described in [39], DNS spoofing could be used to set up a man-in-the-middle between one victim and a domain.

However, our use of DNS spoofing aims at blocking traffic towards chosen domains. In the settings of NURSE, the user can add some domains he wants to block to a blocklist. When NURSE intercepts a DNS-response that reports an IP corresponding to a domain, it checks whether this domain matches one of the entries in the blocklist. Entries can be a fully-qualifying domain name, but can also be used to block a domain and all its subdomains. For example, entering *domain.com* in the blocklist would also blocks *analytics.domain.com*. NURSE then automatically replaces every DNS response for these domains with a spoofed response, replacing the real IP addresses with the loopback address `127.0.0.1`. Therefore, whenever the victim device attempts to contact a blocked domain, the DNS responses contain its *localhost* IP causing the device to send the packets to itself.

An illustration of how DNS spoofing prevents a device from contacting a domain is presented in figure 4.4. In this example, the IoT device attempts to contact *example.com* and *example2.com*, with *example2.com* being a domain that is in the blocklist. When contacting *example.com*, our DNS spoofer only forwards the DNS queries and responses. But when *example2.com* is contacted, the DNS spoofer edits the DNS response, causing the IoT device to send packets to the loopback address when attempting to contact *example2.com*.

Blocking domains could be used to prevent an infected device from sending traffic to malicious domain before cleaning the device and removing the malware. However, blocking domains has multiple applications and is more and more common nowadays, as many people try to block advertising domains or trackers for privacy reasons. A famous example of DNS blocking tools is the PI-hole [34]. Thanks to this open-source tool, users can block specific domains, using their own lists, or some blacklists found on the internet. However, these are not easy to set-up: Pi-hole requires to be run in a Docker or on an additional device (for example a Raspberry Pi hence its name) and requires to modify the DNS IP in

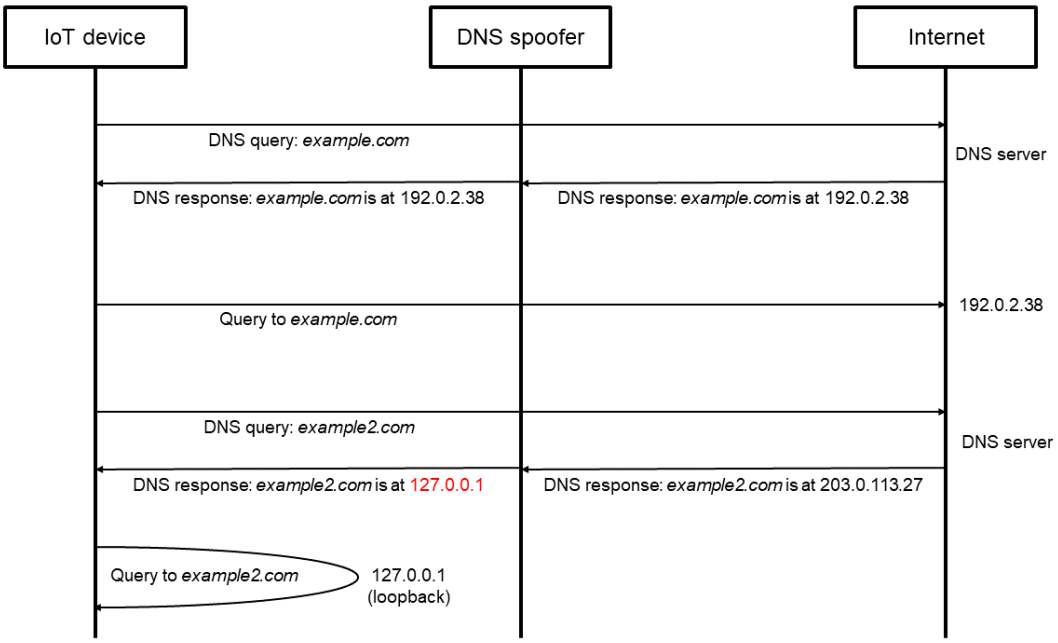


Figure 4.4: An illustration of DNS spoofing

the device parameters.

Therefore, our DNS spoofer could also be used to block unwanted domains, as Pi-hole does. While this is not the main purpose of NURSE, one could download it to use its computer as a domain blocker for other devices in its network, without having to buy additional hardware.

5

Analyzing traffic

5.1. Tool architecture

In the previous chapter, we detailed how we managed to set up traffic interception. Now that we lay in a man-in-the-middle position, we can parse the packets and monitor the traffic features. In this chapter, we detail how we parse the packets, and the features we extract.

NURSE is composed of multiple modules:

- *Host state*: stores settings, the database and manages the child processes
- *ARP spoofer*: performs the ARP spoofing
- *Sniffer*: performs the packet interception
- *Packet Parser*: parses the packet content, to extract data
- *Traffic monitoring*: extracts the flows and information from the packet data, updates the database regularly. This module also contains a classifier to detect suspicious domains that could be generated by Domain Generation Algorithms (DGAs).
- *Traffic Analyzer*: analyzes the database per time window and raises alerts
- *Server*: presents the results to the user via a local web server

NURSE architecture is shown in figure 5.1, with boxes showing the main modules and their purposes, and arrows showing how the data (packets, alerts...) flows between the modules. Figure 5.2 also displays how a packet is analyzed in NURSE. The packet is intercepted with the sniffer and forwarded to keep the connection running. The packet parser then parses the packet header, and sends it the traffic monitor which groups packets per flows. The flows are stores in the host state database, which is then analyzed by the traffic analyzer to detect suspicious events. Finally, the webserver gives the user access to the alerts and to the traffic database.

NURSE uses a multi-threaded approach, as most of these modules have to run simultaneously. The ARP spoofer runs an infinite loop in which it sends ARP spoofing packets regularly to ensure that the ARP spoofing is maintained. Running it as a parallel thread also allows us to send a quit signal from the Host state thread to the ARP spoofing thread when the program is closed. When this signal is received, the ARP spoofer threads undoes the ARP spoofing by sending ARP packets with the actual IP and MAC addresses to all the devices it spoofed. This ensures that the device which were spoofed can find the actual gateway immediately. Otherwise, running NURSE may cause a network disruption until the spoofed devices update their ARP tables and find the real MAC address of the gateway.

Similarly, the host state monitors the activity of each thread and detects ending conditions. For example, it monitors the running time, or the number of captured packets, which allows us to end the process if some ending conditions are met. For example, we allow the user to set up a maximum running time or a maximum number of packets so that the tool automatically closes after a specified time or once a certain amount of packets have been captured.

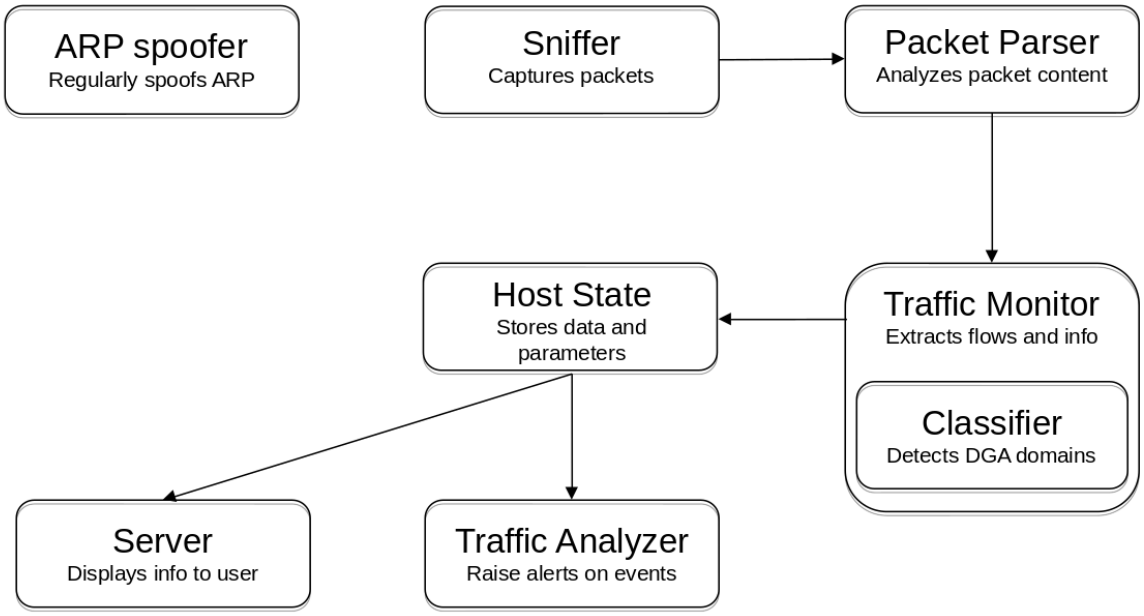


Figure 5.1: Modules and data flows in NURSE

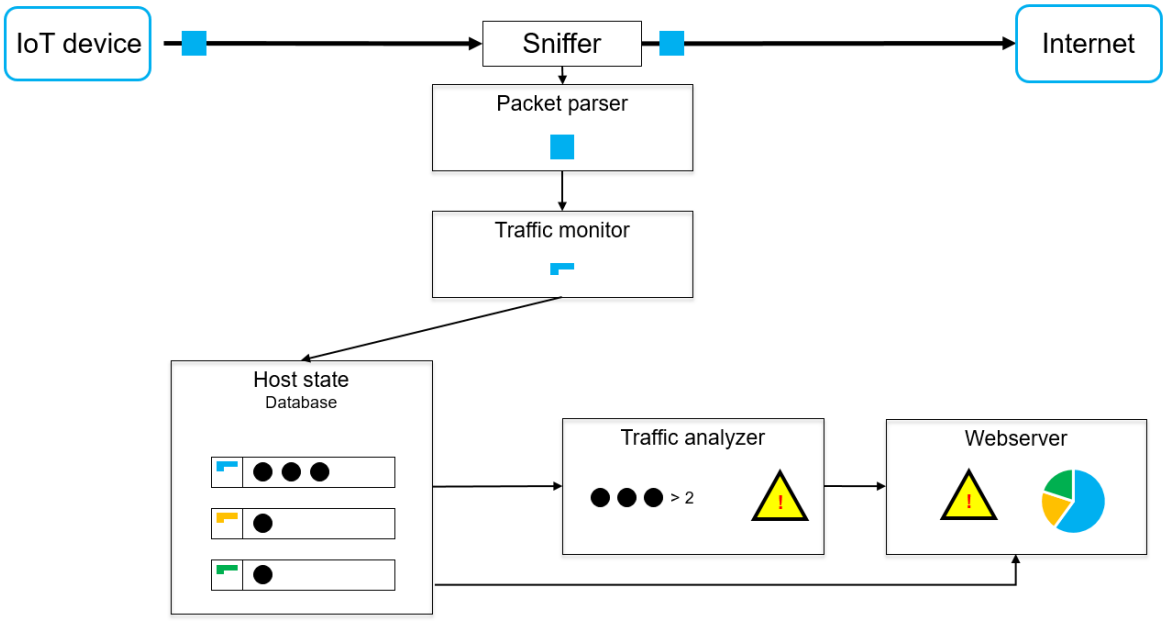


Figure 5.2: Analysis of a packet in NURSE

5.2. Packet parsing

5.2.1. Packet filtering

The packet capture is performed using Scapy [14], which allows our Python script to capture packets at the interface level. Therefore, our sniffer captures all of these packets, but we filter out some as not all of the packets captures at the interface level are relevant for our traffic analysis.

First, we filter the packets based on the Ethernet layer, and more precisely based on the source MAC address. This is to prevent the host from capturing the packets it is sending. The packets sent by the host can be:

- ARP packets sent by the ARP spoofing thread
- Packets that are forwarded from the victim to the gateway to keep the traffic flow uninterrupted despite the man-in-the-middle position
- Actual traffic from the host, for example if the user uses another application while NURSE is running

The first type of packets are generated by NURSE itself and are therefore not relevant to analyze. Similarly, the forwarded packets are redundant because they are sent once we have intercepted the original packet. Therefore, scanning the forwarded packets would cause every packet to be counted twice.

Capturing the third category of packets from our host could prove useful to analyze the traffic of our own host. This could allow NURSE to perform botnet detection on the host it is running by analyzing its outgoing traffic. However, we did not design NURSE with this use case in mind, and the main use case is to monitor other devices than the running host. There exist more specific softwares dedicated to malware detection on the machine such as antiviruses, that also exploit the fact that they are running on the analyzed machine for example by analyzing files looking for malware signatures.

Therefore, the captured packets are from and to the devices that are specified by the user. We drop all other packets, to prevent interference from the running host.

5.2.2. Packet inspection

Flows

Once the packets have been filtered to keep only the ones that are sent or destined to one of the devices we monitor the traffic of, the packets are parsed in the packet parser module. The packets are then parsed depending on their content and the layers identified by Scapy.

First, for each TCP or UDP packet, we extract its *flow* to group packets per flow in our database. A flow consists of

- A source IP
- A destination IP
- A source port
- A destination port
- A protocol: UDP or TCP

With this flow based approach, we can group all packets that go from a port to another. We then keep the timestamp, flags, byte size of the payload and direction (inbound or outbound) for each packet. These features are the ones that are sent to the traffic monitor module to be added to the flow database. As mentioned in section 4.2.3, ARP spoofing does not allow the spoofer to decrypt traffic, which is why we do not collect more general features on the payload, since it can be encrypted.

DNS parsing
In order to collect more precise data, we parse specific protocols which are known to be not encrypted. One of these is DNS. The use of UDP for botnet detection is the idea behind MINIONS project. UDP is interesting because it is an non encrypted protocol, which therefore allows to look at the queried domains and the responses received.

For DNS packets, we parse the DNS payload as well as the UDP header, which allows us both to analyze the flow and the exchanged bytes, and the content of DNS communications. The detection of DNS contents in the packets relies on the flow ports since DNS is recommended to be used with UDP port 53 in RFC1035 [53].

For DNS packets, we parse the DNS responses to extract the answer records. We especially parse the A and AAAA records to obtain the IP addresses associated with the queried domain. Keeping these domain-IP address associations, we build a passive DNS database: a database that keep tracks of IP associated with domains. This database is built live as the tool runs based on the DNS packets intercepted. This allows us to identify which domain are contacted by the device in order to keep track of the contacted domain, and to map the flows IP addresses to domains. Finally, the parsing of the DNS response is where the DNS spoofing detailed in section 4.3 happens. We keep a separate passive DNS database for each host. Firstly, because some devices may receive DNS responses which may be inconsistent, for example if they use a different DNS server. We also keep track of the timestamps of the DNS queries per domain and per device, to keep track of the queries each time they happen instead of merging them per queried domain. Secondly because it makes it possible to get the list of contacted domains per host, which can be presented to the user to show him which domains are contacted by his devices as in IoT-inspector [35].

ARP parsing
As ARP packets are not encrypted, we can also parse their content. We parse these packets to keep track of the MAC and IP addresses of the devices in the network. These associations are stored in an ARP table mapping IP addresses to MAC addresses. This table is stored by the host state and is accessed by our threads when they need to obtain mac addresses of devices. For example, the ARP spoofer is written in such a way that it queries for the MAC address of the targeted devices if it is unknown. However, if a packet that announces the MAC address of a device is intercepted (for example because it was broadcasted to all devices), this association is registered so that the ARP spoofer does not need to query the MAC address again.

DHCP
DHCP is a protocol used in local networks to manage the network. It helps with IP addresses management and for keeping track of which devices are in the local network. DHCP packet parsing can be used to grab the names that are advertised by devices in the local network. This is useful to present some readable names to the user when he looks at the list of detected devices.

In our packet parser, we parse DHCP packets to obtain advertised host names. We then build a database associating the device names to MAC addresses. Combining this with the ARP table allows us to convert an IP address to the advertised host name, which will be useful when presenting data to the user or when listing devices for the selection of monitored devices.

It is important to note that only DNS, DHCP and ARP, which are all unencrypted packets are inspected. This is for two main reasons:

1. inspecting encrypted protocols, for example TLS, is doable in a MITM position as we have, but requires to decrypt and re-encrypt traffic which requires a new certificate to be installed on the spoofed device. As this task requires complex operations to be performed on the device, users may not be allowed or competent enough to install such a certificate. That is why we did not perform deep inspection on the packets

2. performing deep packet inspection on victim devices would reveal all sent and receive data, and not only the unencrypted data and packet headers. This would allow the user to intercept traffic from any device in his house network. While this could allow a user to have a look at the data his IoT devices send to their server, showing what type of personal data is sent to the servers, this would also allow our user to inspect traffic from devices owned by other people in his household, which could cause privacy issues.

5.3. Network monitoring

We mentioned previously that parsing ARP packets could help in monitoring the network to discover new hosts. However, using only ARP can be limited to detect new hosts because hosts usually only advertise their IP address via ARP as they join the network. Therefore, unless the gratuitous ARP packet sent upon joining the network is captured, the only way to discover new hosts via ARP is to perform a live scan of the network. This however is invasive, and may not always be possible. It is also impossible to perform if we want NURSE to be able to parse capture files instead of running live.

Therefore, the packet parser detects when a packet reveals the IP of an unknown device. Of course, we cannot intercept traffic of an unknown device since the ARP spoofer only targets devices that have been selected by the user from the list of detected devices. However, we can detect new devices when a targeted devices communicate with them, and therefore specify their IP as a destination. Upon detection, NURSE sends ARP packets to check if the IP exists, attempt to obtain the MAC address and the device name of this newly detected device and add these to the database, so that the device is now detected and available to monitor for the user. When parsing capture files, this part is disabled, but the newly detected device is automatically added to the list of monitored devices so that the tool parses packets from and to this host.

5.4. Single packet alert detection

Finally, the packet parser also performs some basic checks to detect suspicious activity from a device. As it is parsing the packets headers, it can check their validity before even storing these in the database. For example, the packet parser checks if the source IP address in the IP layer and the MAC address in the Ethernet layer match the associations in the ARP table. The goal is to detect devices that spoof their source IP. In such a situation, the device sends a packet with a source IP address that is not its address, so that the answer to this request is sent to another device. As the device that spoofs the packet will not receive the response, there are few benign applications to such a technique, however, some botnet use IP spoofing to launch or amplify their DDoS attacks. To prevent such attacks, the packet parser checks the source IP field of packets to detects spoofed source IP. In case of a detection, the packet parser directly generates an alert (see section 7.1.1) that is sent to the alert manager of the host state, skipping the data extraction and analysis of the traffic monitor module.

6

Domain scoring

In the previous chapter, we mentioned how the packet parser and traffic monitor could raise alerts on single packets events such as spoofed source IP addresses (see section 5.4). As shown in figure 5.1, the traffic monitoring module which receives data from the packet parser is also coupled with a domain classifier. We describe the domain classification in this chapter.

6.1. Domain generation algorithms

6.1.1. Introduction to DGA

The domain name system is defined in RFC 1035 [53] and is mainly used to convert domain names such as `example.com` to IP addresses. It allows internet users to connect to specific IP addresses without having to memorize the addresses themselves but using domains that may be easier to memorize.

Domain names can be used by attackers for multiple purposes. A first basic example is the use of domain registration as a switch for malware instructions, A famous example of a domain being used as a kill switch is the Wannacry attack. Upon infecting a new machine, the Wannacry ransomware checked if a specific domain was registered before encrypting the files. This domain was discovered by Marcus Hutchins, a security researcher, who registered it, thus preventing the encryption of newly infected machines [36].

As botnet masters want to hide the dedicated IP addresses they use for their C&C servers, to prevent seizure by the authorities, they may use DNS to perform Fast-flux. This technique involves rapidly rotating IP addresses assigned to a domain name. The IP addresses act as proxies for the real command and control server, to prevent detection of the real command and control IP addresses, which prevents authorities from shutting down the botnet.

A third use of DNS for botnets owners is to allow the infected machines to connect to a command and control server to receive orders. As botnet owners want to hide the infrastructure they use to control their botnet, they use a wide range of protocols to communicate with the infected machines. Some involve communicating with machines via IRC [73] or via social media [25], however, these communication channels may be detected and taken down by authorities in a short time. One way to add some unpredictability to command and control servers is to use DNS and to set rendezvous points at random domains using Domain Generation Algorithms (DGAs).

A Domain Generation Algorithm is an algorithm often embed in a malware, that produces a large list of random domains names. The goal is to provide a large list of possible meeting points to prevent them from all being shut down by the authorities before the infected device connect to the command and control server.

To achieve this, the malware runs a domain generation algorithm, usually involving some pseudo-random domain generation, to generate a given number of domains. On his side, the botnet master will generate the same list and choose one of the domains to register and host the C&C server on. Even if authorities can reverse engineer the malware, implement the DGA and obtain the list of domains, they cannot prevent all the generated domains from being registered. They have to wait for the chosen domain to be registered, identify it and shut it down as quick as possible to prevent further communications between the botnet and the C&C. However, the time needed to shut down the domain may be

long enough to allow some of the infected device to receive their instructions.

Therefore, DNS is a protocol that can largely be exploited by botnet masters to manage their botnet and organize communications within the botnet. DGA is the main use of DNS in botnets and is the only one out of the 3 techniques where the DNS exploitation is running on the infected machine (Fast-flux and kill-switches involve domain name management by the botnet master). That is why, we implement a classifier that attempts to detect malicious domains that could be generated by DGA running on an infected machine.

6.1.2. Literature review

DGA domains can be detected based on two approaches:

- The domain registration: tracking the DNS records for the domain
- the domain name itself, identifying patterns generated by DGA

There already exist a lot of DGA classifiers in literature, which often combine these approaches by taking advantage of multiple features.

EXPOSURE

EXPOSURE is a tool designed by researchers from Eurecom, Northeastern university and University of California [13]. This tool relies on DNS features to identify malicious domains. It uses 15 features which are grouped into 4 sets: time-based features, DNS-answer based features, Time To Live (TTL) value based features and Domain Name-based features. These features are obtained by analyzing passive DNS records for the domain, in order to extract records that match this domain. The algorithm takes advantage of the fact that malicious domains are usually short-lived, can have abrupt changes in their access pattern, can be associated with IP addresses spread among many countries, and have short TTL as they rotate quickly. The final domain name-based features are meant to help in measuring the human-readability of a domain, which is essential for websites that are supposed to be used by humans, but is not necessary for malicious domains, which is why some may use many digits or long random character sequences. The tool achieved about 98.5% detection rate in testing conditions. It was also implemented for real-time detection on an ISP DNS server and detected 3117 malicious domains that were previously unknown.

Notos

Notos [8], is a DNS reputation system which evaluates the reputation of a domain based on passive DNS data. It relies on historically associated domains and IP addresses, computing *Related Historic IPs* and *Related Historic Domain* to obtain a set of AS and domains that are related to the analyzed domain. It then relies on 41 features which are separated into 3 groups:

- Network-based features which analyze the BGP prefixes, the autonomous systems linked to the domain and the registrar associated with the IP addresses.
- Zone-based features that analyze the domain name itself: string features analyze the character occurrences and the bigrams and trigrams distributions. They also analyze the distinct TLDs in the related historic domain names.
- Evidence-based features which rely on a malware set captured in a honeypot, and on public IP blacklists.

Building a reputation system out of these features, researchers created Notos, and used it on the traffic of a large ISP. They reached a true positive rate of 96.8% and a false positive rate of 0.38%.

Antonakakis et al. who developed Notos also worked on specific DGA detection using lexical features [9]. In their paper, they use 3 types of features: ngrams based where the median, average and standart deviation of ngrams distributions are used as features, entropy based where they compute the character entropy of second and third level domain, and structural domain features which are measuring the length of domain levels and the number of distinct TLDs in the set of non-existent domains.

Other works

Many other researchers worked on DGA and malicious domain detection. For example Guodong Zhao et al. [77] combine DNS detection with signature and anomaly based detection in network traffic to detect malware infections. The features for DNS detection are similar to EXPOSURE, however, the researchers introduced some new features: active probes to determine whether the domain hosts a web server, and name recognition to detect when a domain uses a famous name (Microsoft, Yahoo, Google), a particular keyword (news, mail, update) or is a phishing name that looks like a famous name to lure users.

Some other works also combined features from previous works [18], often combining two independent approaches, for example lexical and passive DNS features. Finally, some users on GitHub also proposed their own DGA identification models to demonstrate application of machine learning techniques to this problem [7, 31, 33]...

We could have used one of these solutions in NURSE for the DGA detection module, however, we decided to use our own classifier for the following reasons:

- we needed a classifier which implementation was open-source and easy to modify so that we could integrate this classifier in our tool. Most of the previously mentioned solutions did not provide the implementation or provided solutions that were not easy to include.
- some solutions rely on databases, or API which are not compatible with the architecture of NURSE. Since the goal of NURSE is to be downloadable as a software to run on the user computer, one cannot bundle a whole passive DNS database in it. Also, since the goal is to make the tool open-source and easy to use, one cannot use API keys, which would either require to bundle the keys in the software (where they could be stolen) or to ask the final user to get his own API key which is not user-friendly. Also, some databases are licensed which may not allow us to bundle them in our tool.
- our tool needs to be able to classify the domains in real time, so classification time and model weight have to be considered, while these parameters were not always mentioned in previous papers.

For these reasons, most of the features we will use are based on the domain name as a string and do not rely on passive DNS or information that could be queried online such as WHOIS records.

6.2. Classification

6.2.1. Dataset selection

To train a classifier, we needed both malicious and benign domains.

The malicious domain dataset we used is the 360 DGA list from Netlab [50]. We also completed this list with some manually generated domains that were generated using implementations of DGA that were available on GitHub. The manually added DGA families were of 2 different types: DGA that use hashes (from a monero downloader) and DGA which use word lists (gozi). This is to counterbalance the overrepresentation of banjori and emotet DGA in the 360 DGA list, which are random character generation algorithms. The implementations of these algorithms were adapted from implementations found on a github repository dedicated to DGA reverse engineering [1]. The dataset of malicious domains contains about 1500000 domains from 50 DGA families.

There is no benign domain dataset containing random domains which are proven not to be malicious, so building a dataset of domains which are benign is a bit harder. To achieve this, we used two domain lists: the top 1M domains from majestic million [46], and lists of random domains that were uploaded by OpenDNS on GitHub in 2014 [52]. As most of the top domains names are in English, we also get the top 50 domains from each country which are displayed on Alexa [3], to add some linguistic diversity in the training set. While OpenDNS specifies on their GitHub repository that they removed the domains they detected as suspicious from their domain datasets, we cannot be sure that all the top domains are actually benign domains. Therefore, we use the pysafebrowsing [41] package to analyze the domain list with Google Safe Browsing API [27], which allows us to check whether a website can be trusted or not according to Google. Doing so, we find about 1250 malicious domains, most of them coming from the majestic million list and being flagged as `SOCIAL_ENGINEERING` by Google. We

therefore remove these from the benign set, but do not add them to the malicious dataset since they are not linked to malware activities, which we try to flag here.

6.2.2. Feature selection

To select the features we want to use, we implemented some features from previous works and added some of ours, then we selected the ones we wanted to keep based on how well they separate benign from malicious domains and how practical they were for our model. The selected features came from previous works on malicious domain detection [8, 13, 18, 9] in which lexical features linked with length, character frequencies and tri grams were used, and from some publicly available classifiers [7] for the English score feature. One example of an impractical feature was the domain registration. Checking whether the domain is actually registered is a good indicator of whether the domain is benign or is generated by a DGA, since DGA generate a lot of domains of which only one is registered for a short time, while most benign domains are registered to support applications or websites and therefore are meant to stay available for a long time. However, checking whether a domain is registered or not requires to query a DNS server which is too slow and impractical for our tool: this would take up quite some bandwidth from the user of NURSE. Also, such a feature would be redundant since NURSE intercepts the traffic of the IoT device and therefore can intercept a DNS response and check whether the queried domain is registered or not.

Therefore, the features we selected are mostly lexical:

1. Top Level Domain (TLD)
2. total domain length
3. number of subdomains
4. mean of subdomains length
5. consonant ratio
6. consecutive consonant ratio
7. digit ratio
8. repeated characters ratio
9. Shannon's entropy
10. mean of trigram frequencies
11. standard deviation of trigram frequencies
12. English score (ngrams matching with English)
13. word count

For all features except the first two, the TLD is ignored. This is because common TLDs such as *.com* or *.org* can massively influence the character distribution. It would make no sense to consider the TLD in the character distribution of a domain name, since the domain owner usually does not choose the TLD, but only the lower domains.

As some Top Level Domains (TLDs) may be more regulated than others, we quite often see some DGA use TLDs that are uncommon for websites. For example, a DGA for a monerodownloader malware uses a list of 5 TLDs which are *.org*, *.tickets*, *.blackfriday*, *.hosting* and *.feedback* of which the last four are quite uncommon for benign domains. The TLD feature aims at flagging the TLDs that may be suspicious as they host some domains generated by DGAs.

The features 2 to 4 are meant to reflect the domain length, which is one aspect of its complexity. Benign domains, which are designed for humans, should not be too long to be easily memorized and to limit typographical errors when users type the domain name. On the contrary, DGAs do not care about the domain length since the domain is generated by a script.

Features 5 to 7 are meant to measure the character diversity in the domain. As domain names often have to be readable, the cannot contain too much consonants, or consecutive consonants. Digits are

	Naive Bayes	Decision tree	Random forest	Neural network	Nearest neighbors
Training time	0.689	23.770	100.472	368.394	599.492
Prediction time	0.183	0.226	4.066	0.700	151.487
Accuracy	0.874	0.961	0.969	0.942	0.966
F1 score	0.893	0.966	0.973	0.949	0.970
Recall	0.915	0.968	0.969	0.942	0.969

Table 6.1: Comparison of classifiers

also quite rare in benign domains, while they can be used in DGA to add more characters in the pool and therefore provide more variations.

Features 8 and 9 measure character repetition. Domain names are often derived from words in a language or have to be pronounceable, therefore, they follow some basic linguistic properties of the language, while completely random strings do not. Therefore, one can measure the entropy of a string to check if it is closer to English or to a random string.

Features 10 and 11 are also meant to compare the character distribution of domains. Here, we note how much 3 grams tend to repeat themselves in a domain. This is both to detect repeating pattern in a domain, which could be caused by repeating words in some benign domains, or by malicious domains generated by DGA that use some word blocks that they concatenate randomly.

Finally, features 12 and 13 are meant to compare the domain name with English dictionary. For feature 13, we use the python package wordninja [42], which tries to separate a string into words. This is really useful because most domains used for human services are a concatenation of words to be easier to remember. However, random strings generated by DGA are very unlikely to generate words, which could lead in the word count being high since the package would return each letter or syllable as a word. DGA that use digits could generate even more words since digits are split separately from letters, which could lead to domains with more than 12 words as shown in figure 6.1. The English score, is a metric we designed inspired by andrewaeva's DGA detection on GitHub [7]. For each ngram of the domain name (with n between 3 and 5), it counts the occurrences of this ngram in the English dictionary. Then we sum all occurrences for the domain, and take the log of this value to limit the influence of the domain length on this metric. The goal is to give a high score to domains which have ngrams that are common in English, and a poor score to domains that match very little to none English words. The implementation of this English score relies on a CountVectorizer which is a useful counter from the machine learning python package *sklearn* [56].

In figure 6.1, we present the distribution of domains for features 2 to 13 for benign and malicious domains. This shows that even if the distributions overlap, which shows that some malicious domains may be confused with benign domain, the features we selected split the classes. For example, the word count reveals that benign domains mostly contain 1,2 or 3 words, while most malicious domains have more than 5. Figure A.1 also displays the correlation between features. This reveals that except for some features that are linked with each other (such as 2 and 4 since domains rarely have a lot of very short subdomains), the features seem to be independent from each other. We simply note that the length and subdomain length is correlated with many lexical and statistical features, which is normal given that the longer the domain, the more complex it is likely to be, which therefore increases the features linked to character distribution and repetition such as entropy, 3 grams distribution or repetition ratio.

6.2.3. Model selection

For this classification task, we compare 5 models. The comparison is shown in table 6.1. We note that the decision tree, random forest and nearest neighbors have the best performance in terms of accuracy and recall.

The nearest neighbor model performances are comparable to random forest and decision tree, however, its training and prediction time are too long. The training time is not too much of an issue, since 600s are acceptable if we simply have to build the model once and package it with the software, the prediction time is higher than the time for random forests and decision tree. Since we do not want the classifier to slow the tool, we do not keep the nearest neighbors classifier.

Decision trees and random forest have similar results. The training and prediction time are higher

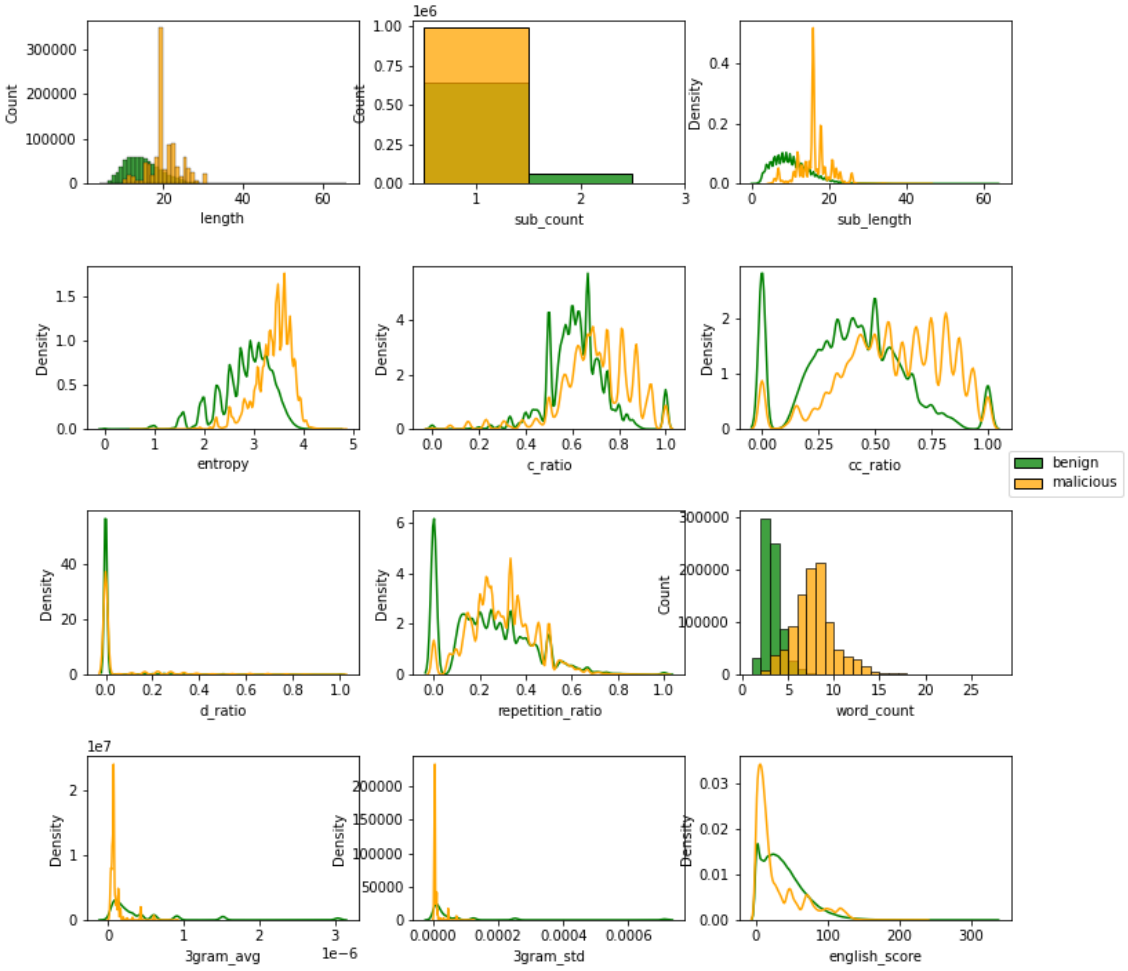


Figure 6.1: Density distribution of training domains for selected features

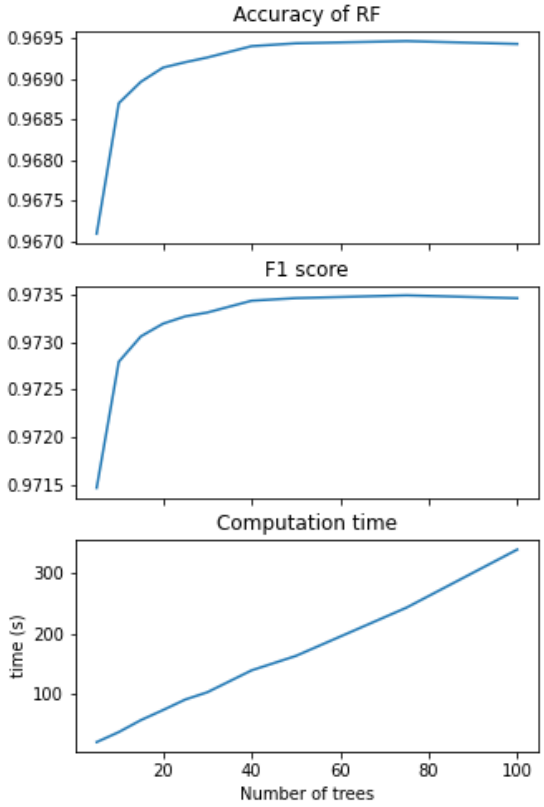


Figure 6.2: Comparison of random forest classifiers with different number of trees

for the random forest, but still acceptable. Since there is a small performance improvement with the random forest, we pick a random forest as the classification model.

Random forests can also be tweaked by changing the number of trees to adapt the complexity and try to improve the accuracy. Figure 6.2 shows a comparison of random forest classifiers based on the number of decision trees used. The graph shows that from 20 trees, the gains in accuracy and f1-score are slim, while the fitting time on training data grows linearly. Therefore, we chose to use a random forest classifier with 20 trees, which allows us to train a classifier in about 2 minutes with no major loss in terms of accuracy.

6.2.4. Results

The confusion matrix of our classifier is shown in figure 6.3. The accuracy of the classifier is about 96.9%. This is a satisfying result considering that we had to limit the features to only lexical features. We however note that the false positive ratio is 3.04%, which could cause some benign domains to be classified as malicious when the tool is used by users. We will try to limit the impact of false positives when designing the rules for DGA alerts in section 7.2.4.

Therefore, while the tool can be used to give an analysis of the domain, it cannot be used as a blocklist to apply on the user's traffic, as it could block legitimate websites and degrade user experience. For example, we noticed when testing the tool that the domain *media.radiofrance-podcast.net* (which hosts podcasts from the French public service radio broadcaster Radio France), was attributed a probability of being malicious of 0.609 and was therefore classified as malicious even if it is a benign domain. This could be related to the fact that the *.net* TLD is often used in malicious domains (see table below), but other features such as the English score should prevent it from being misclassified as the domain name contains French words that also exist in English (*media*, *podcast*, *radio*).

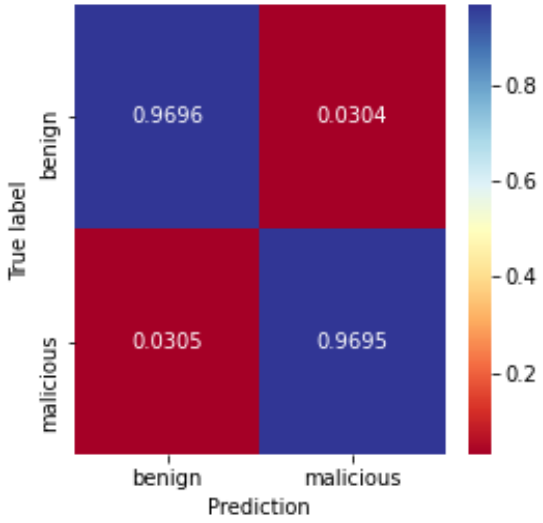


Figure 6.3: Confusion matrix of our domain classifier

DGA family	Accuracy	Train count	Test count	Domain example
matsnu	0.029	590	240	pizzaauthor-brush.com
suppobox	0.090	1504	557	meatgrown.net
bigviktor	0.125	643	264	happenvegetablegeneral.org
mydoom	0.5	34	10	pmqwrehsss.in
virut	0.562	6343	2462	fhifpr.com
...				
gspy	1	66	22	02dce21597c6e71f.net
dyre	1	645	257	f87fd860bb3eda93ef54f9d67025cde561.so
fobber_v1	1	193	70	nuxbrgcdcpogpfxuw.net
omexo	1	29	4	eef795a4eddaf1e7bd79212acc9dde16.net
bamital	1	66	25	5aa591224e42f77e3413118dec43dad7.org

Table 6.2: Classification results per DGA family

As we use a Random Forest, we can have some insights on how the model works. For example, we can compute the feature importance in each tree. In figure 6.4, we present the feature importance and their variation across the 20 trees in the forest. It shows that the main features are the word count and the English score. This means that our model mostly rely on the similarity with English.

This is illustrated when looking at the accuracy on each DGA family. We presented the families for which the detection is the most and the less accurate in table 6.2. We note that while some domains are perfectly detected, some DGA families are hard to detect and the matsnu, suppobox and bigviktor DGA families are almost undetected. Looking at the train and test count, we see that the poorly detected families had enough samples in the training dataset, and that this is not due to a poor representation these DGA families among the training data.

These domains are harder to detect as their generation algorithm uses actual English words that they concatenate. This can be seen looking at the example of a generated domain in table 6.2 or by studying the reverse-engineered implementation of these algorithms for example suppobox and matsnu which use word lists [1, 7]. Therefore, features relying on character distribution or word count give similar values for these domains and normal domains that contain English words, which makes the word list based DGA domains harder to detect. On the contrary, bamital generates domains using MD5 hashes [16]. These are quite easy to identify since they have a fixed length which is unusually long for a domain name, and use hexadecimal characters, which cause a high digit rate and a character distribution that is very different from English as only letters from `a` to `f` are used.

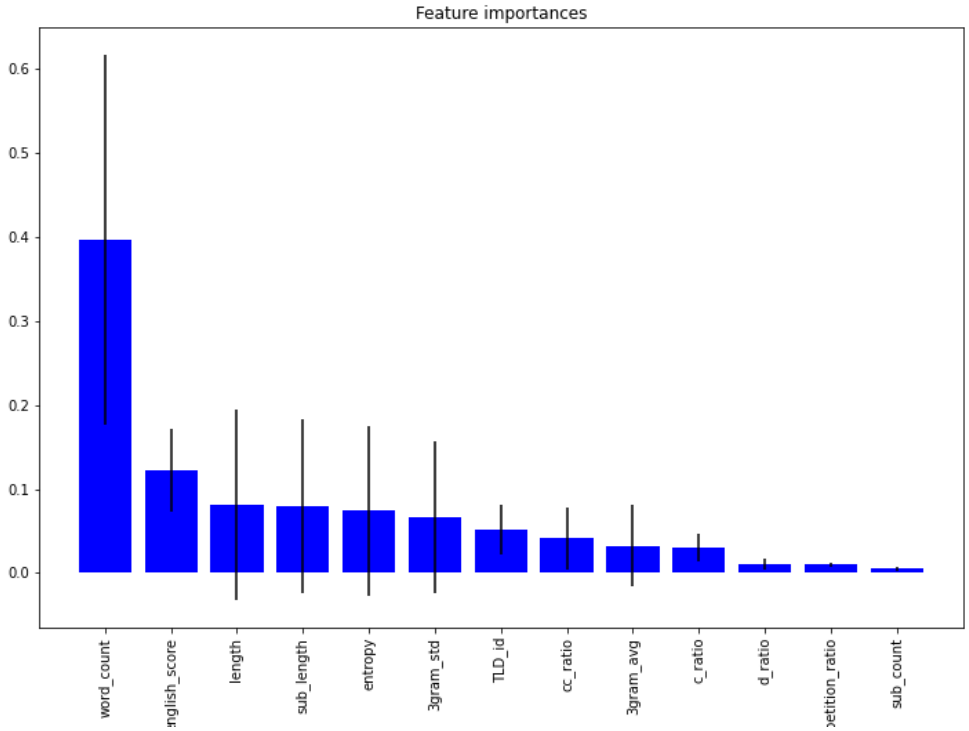


Figure 6.4: Feature importance in our classifier

The DGAs that use word lists are hard to detect based on lexical features. They use English words, which ensures that the character distributions of the generated domains are similar to English. They also use some popular words so it would make no sense to flag these words or the words association as these may also be used in benign domains.

The 96.9% accuracy is satisfying for our classifier, but could be the result of an overfitting of our classifier on the DGA families that were present in the dataset. In that case, the classifier would be very efficient in detecting domains generated by algorithms that were present in the training set, but would fail to detect domains from DGAs that were not used in the training set. This could be an issue as it is very unlikely that the training data included all known DGA algorithms. Overfitting on DGA that were present in the dataset would also cause our classifier to be obsolete soon, as new botnet developers may develop new domain generation algorithm which would not be in the training data and could evade detection.

To determine whether our classifier is overfit, we use a second DGA dataset [12] which was not used in the training. This dataset contained 846616 domains generated by 34 DGA families of which 16 were not in the training data. However, 536226 (63%) domains of this dataset were also present in the training data. To test our classifier, we remove these domains and only consider the domains that were not in the training dataset. Therefore, this evaluation dataset contains 310390 domains, of which 139303 are generated by DGA that were not present in the training data.

The accuracy of our classifier on these new malicious domains is 94%. The fact that this accuracy is lower than the one obtained on the testing data shows that the classifier did not perform as well on the new DGA families as it did on the one it was trained on, which was expected. However, considering that about 45% of the domains in this dataset came from DGA families that were not in the training data, the drop in accuracy is reasonable and shows that most of the new domain generation algorithms are detected.

6.3. Integration in our detection tool

Domain generation algorithms allow the device to communicate with the command and control servers via an unpredictable meeting point. These algorithms generate a lot of domain names from a seed. The botnet owner simply has to register one of the generated domains so that the infected machines connect to it.

The use of DGA can be flagged in two ways:

- by noticing that the devices contacts a lot of non-existent domains. In fact, the DGA will generate a lot of domains and the malware owner will only register one of these, therefore, the infected device will query many non-existent domains before finding the registered one. That is why we will raise an alert when a device received a lot of NXDOMAIN (non existent) DNS responses.
- by analyzing the domains that are queried. As the domains are generated from an algorithm, they can follow a pattern which could be identified. This is why we introduced our domain classifier, which goal is to identify domains that could be generated by DGA.

The classifier is integrated to the tool as a domain scorer. Given a domain name intercepted in the traffic (in a DNS query), we compute its features and use our random forest classifier to compute the probability of being malicious. This probability is multiplied by 10 to show a score between 0 and 10 to the user and the domain score is logged. The domain list with their scores is displayed to the user which allows him to see which domains are contacted by the device, and how suspicious they look. However, since there may be some false positives, the traffic analyzer analyzes the list of queried domains in time windows and checks if the number of domains with a bad score exceeds a threshold. In that case, an alert for suspicious domains is raised (see section 7.2.4).

The DGA detection is also complemented by the previously mentioned NXDOMAIN alert (see section 7.2.5). Even if a DGA algorithm evades our classifier, it may be detected by the NXDOMAIN alert if it queries too much non-existent domains in a short time window. Therefore, for a DGA algorithm to be unnoticed, it would also have to limit its query rate so that the threshold is not exceeded, which would cause it to search for a longer time before finding the rendezvous point with the C&C.

7

Alert system

This chapter presents the malicious behavior that we detect, and how it is flagged. We consider two type of alerts: alerts that flag suspicious single packets and alerts that flag events over a time window.

7.1. Single packet alerts

These are usually not attacks in themselves, but suspicious details in packets that could reveal that an attack is performed.

- Spoofed source IP
- Contacting an IP with no DNS query (Hard coded IP)
- Contacting a blacklisted IP

7.1.1. Spoofed source IP

The spoofed source IP flags packets in which the source IP address field has been filled with an incorrect address. As described in section 5.4 this alert is raised straight from the packet parsing module. When a device sends a packet from a home network, the expected source address is either the local IP address in the network, or the public internet address if the packet is sent over the internet. If the address in the source IP field is none of these two, the packet is flagged.

IP spoofing is not an attack in itself, and can even be used in legitimate situations, for example to simulate packets coming from multiple hosts with all sent from a single host. Attackers may also use it to bypass IP filtering in a network, however, its main malicious application is DDoS attacks. In DDoS attacks, the attacking device sends many packets but does not intend to receive and deal with the replies. Therefore, DDoS attackers spoof the source IP field so that the victim sends replies to random internet endpoints where these will be discarded, instead of flooding the attacking device with replies.

IP spoofing can also be used in reflective DDoS attacks, which use a third party called reflector (which may not be malicious) to amplify the DDoS attack. Reflective DDoS attacks take advantage of services such as DNS which can generate large replies to small queries. The attacking machine sends a lot of small queries with spoofed source IP fields, to a reflector that hosts a service that generates large replies. The reflector sends the response packets to the source IP it received, thus to the intended victim, flooding the victim with large response packets. This allows an attacker to cause a lot of traffic to be received by its victim without having to send as much traffic.

Some defenses have been proposed against reflective DDoS attacks, mostly on the reflector's side to prevent malicious use of their services [65]. A second approach is to block spoofed IP traffic on the internet, and more and more autonomous systems now drop packets with a spoofed source IP address [15]. However, some autonomous systems are still spoofable, which is why we flag spoofed IP packets in NURSE.

A spoofed IP alert may not be a proof that a device is infected in itself, but multiple alerts of this type, combined with alerts that flag DoS activity could definitely reveal that the device is taking part in DoS attacks, and could therefore be infected with malware.

7.1.2. Hard coded IP

DNS allows the translation of IP addresses into domain names and vice-versa. It allows domain owners to assign memorizable domain names to IP addresses. It can also help in moving services from one host to another since the domain can be reassigned to a new IP.

It is now quite rare to contact an IP address directly due to the ease provided by domain names. However, registering a domain name adds an intermediate in the domain registrar. As mentioned in section 6.1.1, botnet masters may not always easily use domain names because the domain names could be taken down by the registrar or by authorities. That is why, some may use hard coded IP addresses in their malware. This is one of the behaviors we intend to flag with this alert.

Another situation that causes a device to contact a lot of hard coded IP addresses is when malwares propagate. When a botnet attempts to find other vulnerable hosts, for example via port scanning, it usually contacts random endpoints or a list received from its C&C server. In this case, the device contacts a lot of devices directly via their IP address with no previous DNS query, which could be flagged by this alert.

We however note that contacting an IP with no previous DNS query is not a malicious behavior. There could be plenty of usages where a device contacts an endpoint directly via its IP address, for example if no domain was registered for this IP address. Another limitation of this alert is DNS caching. To avoid querying DNS each time a domain is contacted, hosts usually save the DNS responses for a small time when the response is first received. Then, when a domain is queried for the second time, if the domain is already in the DNS cache, the device contacts the saved IP addresses directly. Therefore, if NURSE is launched on a device that has cached some DNS entries, requests to the cached domains will appear to be sent to hard coded IP addresses until the cached DNS responses expire, which would raise some alerts on benign traffic.

7.1.3. Blacklisted IP

The final single packet event we flag is contacting a blacklisted IP. There exist multiple blacklists that contain IP addresses that have been flagged for being associated with malicious activities.

The blocklist we use is the Spamhaus project [69]. This project provides lists of domains and IP addresses that have been detected as senders of bulk email, or involved in other malicious activities. Their goal is mostly to prevent email spamming by blocking the senders IP addresses. However, they also provide the *Spamhaus Exploits Block List (XBL)* which contains IP addresses of hosts that have been infected with malicious third party exploits. We therefore check if the devices we monitor communicate with IP addresses of the XBL database, which could reveal that they are communicating with their C&C server or with other devices in the botnet that have been previously detected as sending bulk email or performing malicious activities.

Spamhaus allows the IP addresses to be checked via a public DNS server. For example the IP address 1.2.3.4 can be checked by sending a DNS query for the domain *4.3.2.1.zen.spamhaus.org*. The returned IP address indicates whether the queried IP address is in Spamhaus blacklists. Therefore, whenever our tool intercepts traffic going towards a new IP address, it checks if this IP address is in Spamhaus blacklists.

Thanks to this, NURSE can detect devices that may be infected by checking if they communicate with malicious IP addresses. This may not be a proof that a device is infected since IP addresses can roll and be re-attributed, but Spamhaus databases are updated frequently which should guarantee that the blacklists are relevant and contain actual malicious IP addresses.

This alert however has three drawbacks:

- It is not perfect since the blacklists only contain known malicious IP addresses. Therefore, a device could be infected and communicate with a C&C server without being flagged if the C&C IP address has not been added in the blacklists.
- This alert relies on a third party. If Spamhaus services are unavailable, this alert cannot be raised. We added a timeout on the querying function to prevent the program from waiting indefinitely in this case. Similarly, as the alert uses an external service, some privacy conscious users may be concerned with the fact that the IP addresses contacted by the monitored devices are sent to Spamhaus.
- Checking an IP simply takes one DNS query to the Spamhaus servers. However, depending on the network speed or the number of IP addresses to check, the blacklist checks can seriously

slow down the program. We observed such a slow down specifically when monitoring devices that performed horizontal port scans. The devices generated thousands of random addresses and checking all of these addresses at Spamhaus servers slowed down the analysis (but it did not cause the tool to miss any other type of alert if it had enough time to complete the analysis).

As the last two reasons may be problematic for users, we offer an option to disable this alert in the configuration panel.

7.2. Temporal alerts

These alerts are events that are detected over a time window. The traffic analyzer runs through the packets of a time window and detects whether these packets match detection rules of malicious events. The rules are mostly based on threshold that represent an acceptable amount of traffic over the time window.

The alerts are the following:

- Denial of Service
- Vertical port scanning
- Horizontal port scanning
- Bruteforce attempt
- NXDOMAIN rate
- DGA domains

We pick a time window of 1 minute. This time window is to ensure that short attacks such as small port scan or DGA running with small algorithms can be detected, but is also not too short to avoid generating too many alerts that correspond to the same event. Taking a minute long time window also allows to smooth out the traffic spikes, which prevents from flagging any spike as a DoS attempt. Longer time windows could be chosen, but could be exploited by botnets to avoid detection, for example by running an attack for only half the window, which could still disturb the targeted service. The time window duration and thresholds are initially set in the configuration file, they can be modified by the user in the settings panel of the application.

The thresholds presented for each alert were set based on observation of benign and malicious traffic. For this task, we used captures from malwares captured in a honeypot and benign traffic from IoT devices [58, 4, 67]. We took care of not using captures that would be used for evaluating the detection quality later.

Figure 7.1 present how the analyzer splits into time windows and counts the packets or events within each window. It then compares the count of events within each window with the threshold (set to 4 in this example for simplicity). The figure displays 3 examples. In figure 7.1a, the traffic is normal and the threshold is never exceeded, leading to no alerts being raised. In figure 7.1b, a denial of service attack is happening (labeled with the red dots) and alerts are raised for each time window in which the threshold is exceeded. Finally, the figure 7.1c shows how normal traffic could exceed the threshold in a time window and therefore raise an alert that is a false positive.

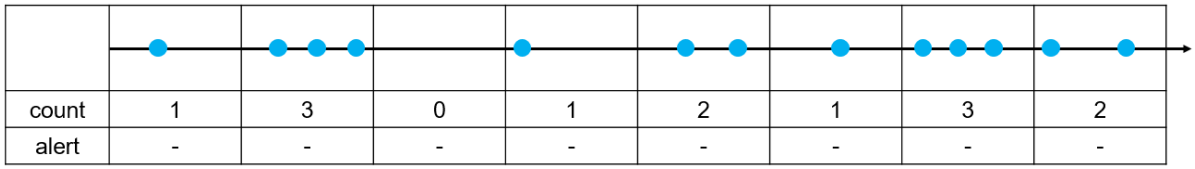
7.2.1. Denial of Service

To detect denial of service attacks, we set a threshold of maximum connections towards a single port of an IP address within a time window.

The analyzer gathers all opened connections per flow key (flows are as described in section 5.2.2). It then the amount of connections opened in each time window. For TCP packets, the connection count is obtained by counting the number of SYN packets sent to the port. For UDP packets, the number of connections opened is simply the number of packets sent since UDP is a connectionless protocol.

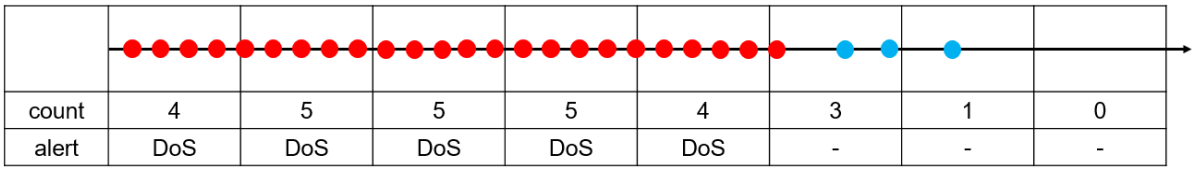
The opened connections are then gathered per key depending on the event we target. Since a denial of service corresponds to a single host opening a lot of connections with a targeted port on a single host, the counts are gathered with keys: (source IP, destination IP, destination port).

Then the counts per time window are compared with a threshold that corresponds to a suspiciously high number of connections for a time window. This threshold has a default value of 120 packets per minute which was set based on observations from malware captures.



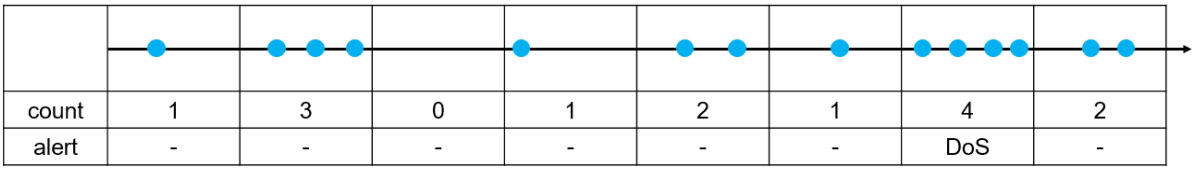
count	1	3	0	1	2	1	3	2
alert	-	-	-	-	-	-	-	-

(a) Analysis of normal traffic with no alerts raised



count	4	5	5	5	4	3	1	0
alert	DoS	DoS	DoS	DoS	DoS	-	-	-

(b) Alerts raised during a DoS attack



count	1	3	0	1	2	1	4	2
alert	-	-	-	-	-	-	DoS	-

(c) Analysis of normal traffic with a false positive

Figure 7.1: Example of traffic analysis (with threshold set to 4)

7.2.2. Vertical and Horizontal Port scanning

The detection of port scanning follows the same implementation as the denial of service, except that it gathers connection with different keys. For horizontal port scanning, flows are gathered with keys (source IP, destination port), for vertical port scanning, the keys are (source IP, destination IP). If the count per key exceeds 30 per minute, an alert for port scanning is raised.

An important point is that horizontal port scanning has to whitelist some services. Otherwise, a user connecting to multiple websites would be flagged as connecting to too many IP addresses on port 80 (HTTP) or 443 (HTTPS). We therefore add a whitelist of ports to avoid raising alerts on these ports. No other ports are added to this whitelist as based on the captures we observed, IoT devices often communicate via HTTP. Some devices may use more specific protocols and other ports, but these are not likely to be used by many different endpoints, which should not raise horizontal port scans alerts. For users who note that their device uses a specific protocol a lot and have port scans alerts raised on normal traffic, we offer the possibility to add custom ports to the ports whitelist.

7.2.3. Bruteforce attempts

Bruteforce is the act of attempting to break into a password protected system by testing many passwords possibilities hoping to find the correct one. This technique is used by malwares to infect devices with poor passwords on connections to services such as SSH or Telnet.

To detect bruteforcing, we use the same approach as in denial of service detection, except that the destination ports are merged if they correspond to SSH or telnet. The goal is to detect multiple connections that would correspond to multiple passwords guesses. The threshold is set so that a device that opens 5 connections in a minute is flagged as bruteforcing the service. Even considering that SSH allows 6 authentication failures before disconnecting [71], bruteforcing while evading the connection would therefore only reach 30 guesses per minute, which is slow for bruteforcing.

This technique only relies on the number of connections. More precise approaches can be taken, especially for Telnet which is not encrypted, but would require to analyze the packets content which could be invasive and would only be possible for non-encrypted protocols. It can however be noted that some advanced network monitoring tools attempt to detect SSH bruteforcing in encrypted traffic [70]. We however did not implement this technique as it only works in specific cases and requires monitoring the connection status precisely.

7.2.4. DGA domains

We already detailed how DGA domains are detected thanks to a classifier in chapter 6. We however, do not raise an alert immediately when a domain is flagged by the classifier, instead DGA domains are set up as a temporal alert to prevent false positives.

As the classifier is not perfect and may classify benign domains as false positives, we do not raise an alert for each domain classified as DGA. A temporal approach also suits DGA detection because DGA are designed to contact many domains in a short amount of time to find the rendezvous point. That is why, a machine running a domain generation algorithm would be flagged even with a temporal alert. To prevent the DGA from being successful or evading detection, we set the threshold at a low number of domains per time window. The default value is 5 domains flagged as malicious per minute. This prevents false DGA alerts as it would require that the classifier raises 5 false positives in less than a minute. However, it also makes it harder to evade detection, as 5 domains per minute means that the DGA should take at least 12 seconds per domain, which is a slow rate and could cause the DGA to find the selected domain in a long time time, which partially defeats the purpose of DGA.

7.2.5. NXDOMAIN rate

This alert is raised when one host sends a number of DNS queries that returned a NXDOMAIN (non-existent domain) code that exceeds a certain threshold. This alert complements the alert for malicious domains and allows to flag DGA even if the classifier misses the domains as malicious, since the host that runs the DGA will certainly contact many non-existing domains before finding the actual meeting point.

However, generating NXDOMAIN is not always due to malware, they could be generated by a user making a typographical error in a domain, or to check that the domain name server is working correctly, as does the Chrome browser on startup to detect DNS hijacking [20]. The chosen threshold is 5 NXDOMAIN per minute, which should prevent some of these false positives since Chrome only queries 3 non-existent domains on startup.

8

Domain whitelisting

8.1. Need for domain whitelisting

The alert system we detailed in chapter 7 may not be precise enough. In section 10.1.2, we will detail the detection results and show the limitations of our system. In this evaluation, we note that normal traffic from some devices may raise false positives.

For example, some Google devices (Google home, Android devices) are flagged because they create too many connections to a host. In fact, they use the QUIC protocol developed by Google which goes over UDP and creates a lot of small packets, which could exceed some of the thresholds we set and therefore lead to DoS alerts.

One possibility to prevent these false positives could be to simply ignore thresholds for the QUIC protocol. This could fix the false positive issue, however, NURSE would also miss DoS attacks performed on QUIC protocol, thus increasing the false negative rate. While QUIC attacks are not common, as few domains use the QUIC protocol, it could be more and more used in the future, and QUIC DoS attacks could happen more and more often.

Therefore, another approach could be to consider that when Google devices contact the domains they communicate with in QUIC, these communications are not part of attacks. In this approach, we take a list of domains over which alerts should not be raised.

In this chapter, we present some methods we explored to identify the domains to whitelist. The goal is to identify the domains that are contacted by IoT devices in their normal use. We also look for domains that are only dedicated to IoT devices. Domains that host infrastructure dedicated for human users could also be targeted by DDoS attacks to disrupt the human-facing service. Of course, IoT infrastructure may also be attacked in DDoS attacks, but is generally built to be able to serve content for many devices and may be more resilient to such attacks.

8.2. Profiles of devices

Detecting IoT domains is a task that is useful for IoT device identification, as we detailed in 2.2.2. Some researchers built IoT-identification techniques that identified the device model or manufacturer based on the domains they contacted. Therefore, they built some domain profiles for each model to help with identification. IoT-inspector, which we mentioned in section 2.3, was also designed to gather a crowd-sourced dataset of such IoT profiles.

We could simply take the domains used in these profiles that are specific to IoT devices and use them as IoT domain lists for our whitelist. Even if this approach ensures that the domains are really contacted by IoT devices and therefore host IoT infrastructure, it requires to buy the device to build such a profile. Thus, it is impossible to build an extensive domain list with this method since this would require us to buy all IoT device models that exist.

Therefore, our goal is to find domains dedicated to IoT infrastructure without having to own all the devices that use this infrastructure. However, as more and more datasets of IoT traffic captures and domain lists are released, we consider using these as a starting point, hoping to find domains that were not listed in the original data. For our research, we will use two datasets containing traffic captures of multiple IoT devices from multiple manufacturers [58, 67].

We parsed the captures and extracted all DNS queries. For each device, we listed the contacted domains and ranked them by number of DNS queries. We present some of these profiles in listing 8.1. From these profiles, we note that devices seem to contact multiple types of domains:

- Generic domains such as *www.example.com* which is likely to be contacted by the Amazon Echo for connectivity checks
- Some domains dedicated to IoT devices on external services: for example, we note that both the Amazon Echo and Netatmo Welcome contacted subdomains of *ntp.org* which is used to synchronize clocks. Similarly, we see that the Amazon Echo contacts *www.meethue.com* probably to interact with a smart bulb from Philips in the house.
- Domains that are from the manufacturer infrastructure, which can be identified as the second level domain has the name of the manufacturer. We then observe that among these domains, some are subdomains that could be dedicated to IoT devices: *api*, *apicom*, *softwareupdates*, *device-metrics*...

Now that we have identified some IoT related domains from the manufacturer infrastructure, our goal is to find more of these domains that may not have been seen in the traffic captures.


```
{
    "Amazon Echo": {
        "www.example.com.": 1793,
        "www.example.net.": 1793,
        "www.example.org.": 1791,
        "device-metrics-us.amazon.com.": 675,
        "amzdigitaldownloads.edgesuite.net.": 299,
        "pindorama.amazon.com.": 120,
        "www.meethue.com.": 118,
        "softwareupdates.amazon.com.": 10,
        "todo-ta-g7g.amazon.com.": 4,
        "pins.amazon.com.": 2,
        "1.north-america.pool.ntp.org.": 2,
        "2.north-america.pool.ntp.org.": 2,
        "3.north-america.pool.ntp.org.": 2,
        "ntp-g7g.amazon.com.": 1,
        "0.north-america.pool.ntp.org.": 1,
        "det-ta-g7g.amazon.com.": 1
      },
      "Netatmo Welcome": {
        "apicom.netatmo.net.": 223,
        "vpn.netatmo.net.": 10,
        "v3.netatmo.net.": 10,
        "v0.netatmo.net.": 10,
        "v5.netatmo.net.": 10,
        "v4.netatmo.net.": 10,
        "v6.netatmo.net.": 10,
        "v2.netatmo.net.": 10,
        "v1.netatmo.net.": 10,
        "clients3.google.com.": 9,
        "2.android.pool.ntp.org.": 7
      },
      "PIX-STAR Photo-frame": {
        "api.pix-star.com.": 625,
        "iptime.pix-star.com.": 5
      }
}
```


Listing 8.1: Profiles of IoT devices

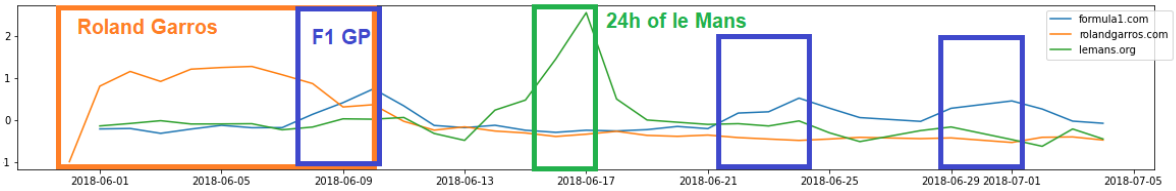


Figure 8.1: Passive DNS counts for sport related domains over June 2018

8.3. Finding other domains

8.3.1. Passive DNS counts

A first approach to find IoT domains is inspired by the article *A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild* [62] in which researchers noted that Alexa and Samsung devices have daily patterns that relate to humans activity. Therefore, an hourly analysis of queries to some domains could maybe reveal domains that are related to IoT activities. Some examples of noticeable behaviors could be:

- patterns that follow human activity as the devices are mostly used when humans are aware, and therefore generate more traffic during these periods
- regular pings to some domains for devices that check if new updates have been released on a regular basis
- traffic spikes in the few hours after a new firmware update has been released, as all the devices will have to download it.

We had access to a passive DNS database [63], in which DNS records were saved. One advantage of passive DNS over active domain scanning is that we are not actively scanning the company infrastructure, contrary to other domain discovery techniques. The passive DNS data also provides the count of how many times each record has been seen over the recording period. Our intuition was that these counts could reflect the activity towards these domains even if DNS queries the reflection may not be perfect as some devices may have cached DNS records.

Unfortunately, the database we had access to aggregated these counts per day, which smoothed out the daily behavior and made it impossible to observe the patterns we listed previously. We did not observe significant daily variations for IoT domains that we had found in device profiles.

We however attempted to confirm our intuition that DNS counts were correlated with activity. As we could not study hourly patterns, we looked for events that happened on precise days, to check if the DNS counts were affected on these days. In figure 8.1, we plot the passive DNS counts of domains related with sport events that happened in June 2018 and note that:

- *lemans.org*, website of the 24 hours of Le Mans (an automobile endurance race) saw a spike in pDNS counts on the weekend of June 16th and 17th, during which the 2018 edition of the 24 hours of Le Mans happened.
- *rolandgarros.com* website of the French Open (a tennis tournament) had a decrease in traffic after the 10th of June. In 2018, the tournament was held from May 27 to June 10.
- *formula1.com* had some traffic spikes on the weekends where Formula 1 Grand Prix were held.

These observations show that the passive DNS counts observed for domains dedicated to these events correlates with the public interest for these events, which backs our assumption that passive DNS counts can be used to study traffic patterns via passive DNS counts.

8.3.2. Passive DNS analysis

Distance between IP spaces

We now explore ways to discover new IoT domains using passive DNS data. To help in discovering such domains, we take some domains that we already identified as IoT domains in the device profiles.

Subnet mask	Number of IP addresses
/0	$2^{32} = 4294967296$
/4	$2^{28} = 268435456$
/8	$2^{24} = 16777216$
/16	$2^{16} = 65536$
/20	$2^{12} = 4096$
/24	$2^{8} = 256$
/28	$2^{4} = 16$
/32	$2^{0} = 1$

Table 8.1: Number of IP addresses per subnet mask

We first query the passive DNS data to obtain all the subdomains of the manufacturer domain. Then, we attempt to identify which of these domains share some infrastructure. If IoT dedicated domains are stored on a dedicated infrastructure, that is separated from other parts of the manufacturer network, this could help us in identifying IoT domains by looking for domains that are hosted with the known IoT domains.

The distance measure we use is the subnet mask of the union of IP addresses for the two domains. Subnet mask is a characteristic of a network of IP addresses that measures the number of bits in the shared prefix of all IP addresses in this network. This value ranges from 0 to 32 (since IP addresses are stored on 32 bits) and the higher the value, the less IP addresses in the subnet. Table 8.1 presents the number of IP addresses in subnets depending on the mask size. To measure the proximity between two domains, we query the records for each domain, extract the IP addresses for each and merge them in a set. We then look for the smallest subnet mask that includes all IP addresses in the set. The higher the subnet mask, the smallest the network that includes all IP addresses, which means that the domains are stored on the same infrastructure.

Figure 8.2 shows the plotting of the distances for all subdomains of the *withings.net* domain. We can observe that the domains are grouped into multiple subnets. In some captures from Withings devices [67], we noted that baby monitors from Withings contacted *babyws.withings.net*. Looking at domains that are close to this domain, we find domains that seem to be related to IoT or mobile devices: such as *mobile.withings.net* or *scalews.withings.net* which is dedicated to smart scales and health sensors.

We provide more distance matrices in appendix (figures A.2 and A.3) where some domains hosted on the same infrastructure can be noticed in the *lutron.com* and *sonos.com* subdomains. Following some domains that could be related to their IoT infrastructure such as *updates.lutron.com*, we discover many auxiliary subdomains on the same hosts such as the *lutron102.lutron.com*.

We however note that this method may not work for all domains since some manufacturers may rely on dynamic hosts servers or content delivery networks to serve their domains on multiple infrastructures, which would defeat the subnet mask measure since domains may be stored on multiple IP addresses of very different networks.

IP address relation graph

Another way to visualize the connections between subdomains could be to use a graph. In this section, we introduce a graph visualization that helps in finding IoT dedicated domains.

In our first approach, we used a wildcard to query all subdomains of the manufacturer domain. This method revealed the connections between all of these subdomains, however, it did not show the connections with domains that are not directly subdomains of the manufacturer domain.

To build our graph, we pick a starting domain that we identified as an IoT domain from a traffic capture. Then, we get all IP addresses associated with this domain, and for each of these addresses, perform a reverse DNS query to find all domains that are associated with this address. By merging these two steps into one, we create a relationship between two domains. We then continue from the newly found domains until a certain distance from the starting node is reached.

This method is not completely reliable, since IP addresses may be reassigned to other unrelated domains, especially if they are part of shared hosting environment. We however present two manufacturers for which the IoT infrastructure was visible in the graph.

Figure 8.3 presents the graph obtained starting from the domain *babyws.withings.net* (in red) which was found in captures from Withings baby monitors. We observe a cluster on the left with some of

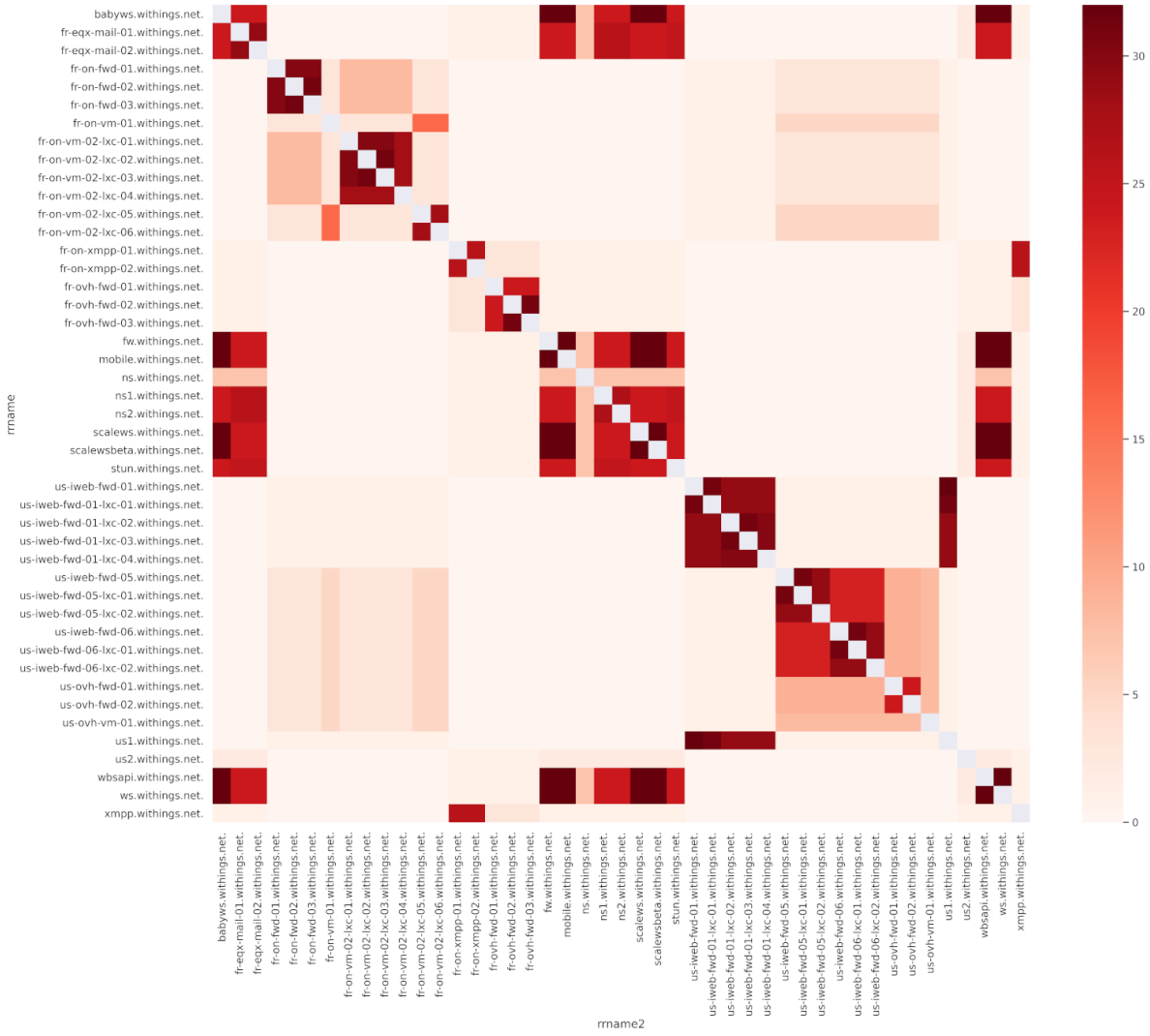


Figure 8.2: Distance between Withings subdomains

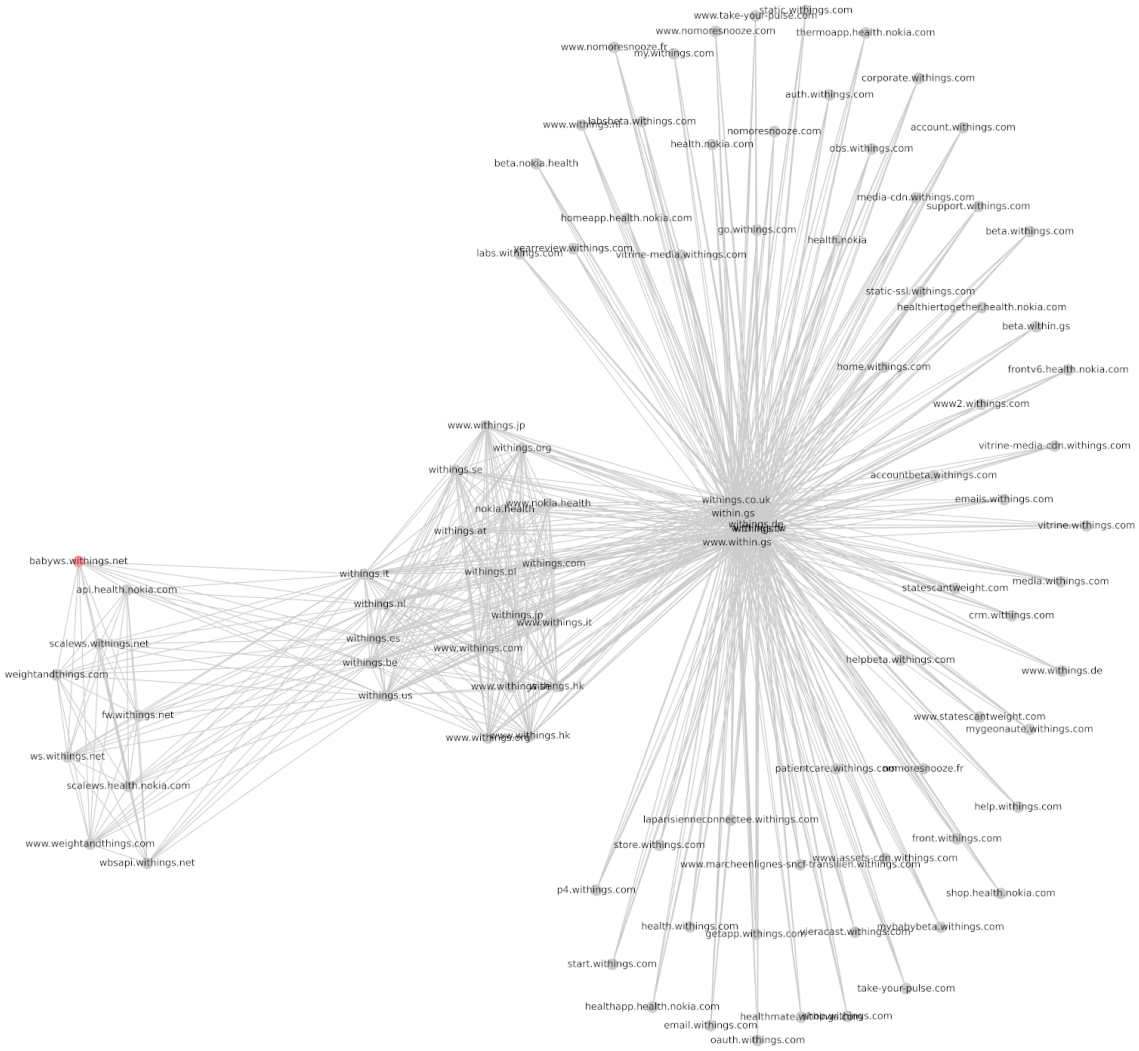


Figure 8.3: Graph of withings domains

the domains we identified in section 8.3.2. However, we also note that some domains in this cluster are from nokia.com which shows that the Nokia health APIs are actually hosted on withings network infrastructure (Withings has been acquired by Nokia in 2017). This method therefore revealed some API endpoints for Nokia devices that had not been found previously.

We then note that the graph goes through domains of Withings with other top-level domains, that we had not considered in the heatmap since we only used subdomains of *withings.net*. However, this bigger group seems to host mostly the human facing domains of Withings, since we see domains with country top level domains, store and promotional domains, support domain...

Similarly, another graph starting from *apicom.netatmo.com* in appendix (figure A.4) reveals some auxiliary domains *f*.netatmo.net* that seem to be hosted with *api.netatmo.net*, which could be dedicated to some IoT devices. However, these domains did not appear in the captures from Netatmo devices we used [67] (see listing 8.1).

The fact that some domains we observe in captures are not shown on the graph could also be due to the temporal evolution of the domains. The captures we used are from a dataset of the University of South-Wales [67] which captured traffic of devices in October 2016. Therefore, as we used passive DNS from 2018 for these visualizations, the domain infrastructure may have evolved between these dates. We observed such an evolution when plotting the graph for Netatmo domains with 2020 data

with the same starting domain. The *f*.netatmo.net* domains had disappeared and the graph was closer to a complete graph which revealed that all domains were stored on the same hosting infrastructure.

8.4. Conclusion for IoT domain census

We proposed multiple approaches to identify IoT dedicated domains that could be added to whitelist to prevent false positives in NURSE detection. We did not perform a full census of IoT dedicated domains as this would be a full research project in itself, that would certainly be related with IoT device identification. But we explored some visualizations that could help in isolating IoT dedicated domains from other domains. We however note that these techniques are highly dependent on the network infrastructure of the targeted domains, and may not work if shared hosting providers are involved in it. The dedicated domains may also disappear or be renamed over time, which would require to regularly check for new domains.

9

Presenting results to the user

9.1. User interface

9.1.1. Detection results

To simplify the use of our botnet detection solution, we decided to provide a graphic user interface. Thanks to this, users do not have to read logs from our tool to see if some alerts were raised.

The webserver is lauched automatically on port 5000 when the tool is started. It can then be accessed from any web browser. It displays the following pages:

- *devices*: in which the user can select the devices they want to track
- *domains*: which lists the domains contacted by tracked devices.
- *alerts*: which lists the alerts that were raised
- *config*: in which the user can change detection settings

The domains page displays the contacted domains as well as the amount of bytes sent and received. This allows the user to

At the moment, the information displayed is quite limited. Studies on user interfaces for such a detection tool could be done to determine the best way to present the collected data. We however have all the necessary data to display more precise information or draw more visualizations as in IoT-inspector [35].

9.1.2. Selection of devices

The selection of monitored devices can be made by changing the configuration files before launching the tool. However, this may be complex for non computer literate users, and requires knowing the IP addresses of the devices to monitor. We therefore moved the device selection to the web application so that users can have a graphic interface to select the devices they want to scan. On this page, they can track a device in one click by ticking the associated checkbox.

For the device page (see figure 9.1), we tried to display some non technical information about the device that could help users in identifying the devices, as users who do not know about networking may not know which IP correspond to each device. Therefore we list the device names obtained as in section 5.2.2, and used *mac2vendor* API [48] to translate MAC addresses into manufacturers in an attempt to get the brand name. This can help users to identify the devices in the list based on the manufacturer name, in case the device name is not explicit enough.

9.1.3. Visualizations

As an example of a visualization that can be displayed on the web application, we designed a map visualization of the countries contacted by devices. This choropleth map as shown in figure 9.2, shows a world map with countries colored depending on the amount of data sent to IP located there. This map allows the user to visualize where packets are sent, and could potentially reveal some suspicious behavior if a device contacts countries that are not supposed to be contacted during its normal activity.

Device list

Here are the devices we found in your network

Last update Mon Jul 5 15:52:04 2021

IP	Device name	Manufacturer	Scanning
192.168.1.1	home	Zyxel Communications	☐
192.168.1.144	[redacted]	Foxconn (Hon Hai Precision Industry)	☐
192.168.1.46	HUAWEI_P10_[redacted]	Huawei Technologies	☐

Figure 9.1: The device list page

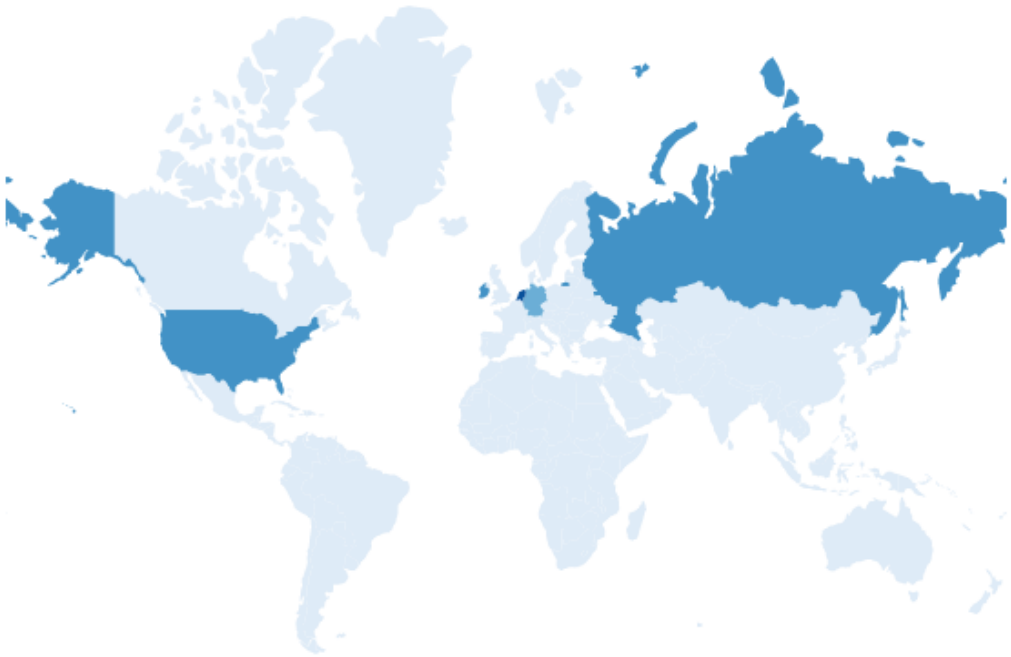


Figure 9.2: A map visualization of contacted domains on an Android device

9.2. Simplicity of use

In section 3.2, we mentioned that the tool we designed was meant to be used by non computer literate users. We therefore tried to make the installation and setup as easy as possible.

The tool internalizes the networking setup to intercept packets automatically. Therefore, the only setup part is the tool installation. On Linux, the tool can be run from the source code as a Python script. It may require some packages which can all be installed with the python package manager. However, admin privileges are required to perform the packet interception.

On Windows, the installation is more complex due to a dependency. We used Scapy [14] for packet interception, which itself requires Npcap to work on Windows. Therefore the user must install Npcap before running NURSE. To ease the installation, we wrote a small script that downloads and launches the Npcap install. It still requires the user to go through the installer and give admin privileges for the installation but automates the download. We however note that the admin privileges are only needed for this installation and will not be needed when running NURSE.

Our tool can then be run from the source code in a Python environment in Windows. However, as users may not know how to run a Python script, we also provide an executable version, compiled with the python package Pyinstaller [74]. This packages generates a folder with embedded packages and a `.exe` file which can simply be run by double-clicking on it, even without a Python environment on the computer.

The installation is not extremely easy, which is mostly due to the fact that we implemented the tool in Python, which requires a Python environment to be executed. We however tried to ease the installation with the executable file on Windows to it usable without requiring the user to run it from command line.

10 Evaluation

10.1. Detection results

10.1.1. Evaluation method

As this research project was run from home due to COVID-19 crisis, we had no available IoT devices and test environment to run malware in safely to test NURSE against real malware live. Therefore, we adapted the architecture to allow parsing of offline traffic captures instead of using live packet capture. This required to change the way components interacted together to allow some to be disabled, for example the ARP spoofer which is not useful if traffic has already been captured.

Another important point to note is that all the tests presented in this section have been performed on a personal laptop with 16GB of RAM. Therefore, the storage was limited and some captures could not be opened and fully stored in RAM when running the program. Therefore, the packets were streamed from the file. This allows the script to run on large files representing long captures, however, the streaming could take more than a day for a single file. Therefore, we introduced a stop rule for long files and stopped the evaluation when 2500000 packets had been streamed. This is indicated with a packet number of ≈2500000 for the corresponding captures. Due to the asynchronous aspect of the tool, and to keep sure that all components stop correctly, the tool does not abruptly ends when the cap is met, hence the approximation and the fact that for some of these scenarios, the number of flagged packets exceeds 2500000.

For the evaluation, we used two datasets, CTU IoT-23 [55] to check botnet attacks, and captures of normal traffic from the VARIOT dataset [61].

The first dataset contains captures of botnet activities running on IoT devices. The dataset includes 3 devices: an Amazon Echo, a Phillips Hue and a Somfy door lock device. The devices are then infected with malwares such as Mirai or Okiru, and traffic is captured. The researchers then labeled the flows manually indicating whether these were parts of horizontal scans, attacks, C&C communications, DDoS file downloads, benign trafic... We used these flows classification to label scenarios with the major type of malicious traffic that can be observed in the capture. These labels are presented in the column ”Type” of table B.1.

The second dataset contains captures of compromised devices and normal traffic. It is important to note that even if a few captures correspond to DoS attacks, and firmware changes which could be relevant for our botnet detection tool, most captures correspond to an attacker controlling the device remotely. In these captures, an attacker controls a device via an exploit, and performs basic actions. For example, when attacking a smart bulb, the attacker can turn the device off, adjust the brightness... We note that while these attacks are not benign, they are not botnet attacks and are mostly involve normal actions, except that these are triggered remotely by the attacker. For this reason, we do not evaluate NURSE on the captures of compromised devices in VARIOT dataset, but we use the captures of benign traffic to check if alerts are raised on non-infected devices.

10.1.2. Results analysis

In this section, we present the results of our botnet detection method. We first analyze the results in a detailed way, looking at the alerts raised for each scenario in order to better understand which scenarios

raised alerts and which did not. We then try to elaborate a decision process to simply label the traffic captures as *benign* or *malicious* traffic, and evaluate NURSE on its precision and recall.

10.1.3. Detailed results analysis

Table B.1 presents the alerts raised on the scenarios of the IoT-23 dataset. The alerts that are not present in the table were not raised during the analysis. We first note that most of the horizontal scans and DDoS events have caused the corresponding alerts to be raised which shows that detection for these events seems to work on real attacks. Therefore, NURSE seem to be able to detect these real botnet attacks.

However, we note that some scenarios such as *CTU-Malware-20* and *CTU-Malware-21* with malicious traffic did not raise any alerts. These scenarios only contain communications with a C&C server. Looking in detail at the results, we note that the malicious domain that was contacted *top.haletteompson.com* was not flagged as DGA domain and that the corresponding IP was not blacklisted. The communications then happened over port 443 and were encrypted using TLS. As we cannot decrypt communications in ARP spoofing to perform deep packet inspection, we had no possibility to detect that this communication was malicious if the domain and associated IP were not classified as malicious.

We finally note that no alerts were raised on the benign traffic captures which shows that there were no false positives in this dataset. However, the evaluation on some benign captures of the VARIOT dataset shown in table B.2 reveal that some devices raised alerts in benign traffic.

While most of the devices have almost no alerts raised in benign traffic, we note that some devices from Netatmo, Google and Xiaomi raised DoS alerts on benign traffic. Those alerts were mostly triggered by exchanges over UDP where a lot of small packets were sent in a short time window. Such a behavior for example, is quite common in Google products as Google developed QUIC protocol over UDP to send data. However, it is not possible to completely whitelist UDP exchanges as some botnets may also perform DDoS attacks over UDP.

These false positives show the limitations of our threshold based approach for DoS detection. We could raise the thresholds to ensure that these are not flagged, but this could cause some attacks to be unnoticed.

We however note that the devices that cause the false positives are from major IoT manufacturers. Therefore, performing a census of the domains used by these manufacturers could allow us to whitelist the benign domains that are contacted by these devices, and therefore prevent these false positives. This evaluation is what lead to the idea of the domain census presented in chapter 8. Due to the complexity of this task, and time constraints, we could not perform a full census of these domains and test NURSE again with a domain whitelist obtained in this way. However, we note that we could also simply manually whitelist the domains that caused alerts to be raised on these captures, thus preventing false positives in the future for similar devices.

10.1.4. Performance measurements

Decision threshold

We will now attempt to measure the accuracy of our botnet detection method. However, we currently only have the raw results, with the number of alerts raised per capture. We therefore have to design a decision process to chose whether a device has to be flagged as infected or not based on the number of raised alerts.

The first step to do this is not to consider the number of raised alerts, which may depend on time (since some alerts are computed over time windows) but the number of packets flagged in alerts, as in tables B.1 and B.2.

We then have to decide when to flag a device as infected. We could simply consider that any device raising an alert is infected. However, we noted in chapter 7 that many alerts could have false positives. For example, the alert raised for hard-coded IP addresses is always triggered by some IP addresses that were stored in the DNS cache before we launched the monitoring. Therefore, we limit the impact of the alerts such as hard-coded IP that do not reflect malicious behavior directly, but are more designed to help in detecting when a lot of such IP addresses are contacted, for example in port scans.

For our decision process, we will drop some alerts and only consider the following:

- DoS alerts
- Port scan alerts (vertical and horizontal)

- NXDOMAIN rate alerts
- Domains flagged as malicious by our DGA classifier
- Packets with spoofed source IP field
- Blacklisted IP addresses

Now that we have the number of packets in all of these alerts, we sum these values, and we compare it to the number of intercepted packets, which gives us a ratio. We consider a device to be infected if this ratio is higher than 0.1.

This threshold value is arbitrary, and more extensive research could be conducted to determine the ideal value. However, we picked this value with the following considerations:

- The threshold is picked to prevent devices raising one or two alerts among thousands of packets from being immediately flagged as malicious.
- A Denial of Service attack or a port scan generate a lot of packets, which is unlikely to represent less than 10% of traffic

We however acknowledge that this threshold gives a possibility for evasion to malwares since the malware could slow down its malicious activities and cover them with randomly generated normal traffic to remain undetected. However, just like the thresholds in our DoS and port scan detection rules, such behavior would considerably slow down the attack, which would weaken the botnet and therefore partially mitigate its effects, even if it remains undetected.

Results

With the previously mentioned method, we present the classification results on the CTU dataset in table 10.1 and in table B.3, with a label equal to 1 indicating a scenario flagged as malicious traffic. In the classification column, red values indicate wrong classifications.

With this classification we can now evaluate our classification. We used 83 scenarios, 23 from the IoT-23 dataset, and 60 captures of normal traffic from the VARIOT dataset. The classification results are shown in table 10.2.

From this table we can compute the following metrics:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}} \approx 0.867$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \approx 0.714$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 0.75$$

We note that the precision is about 71.4%, this is caused, as seen in section 10.1.3 by the false positives on some traffic peaks seen with some devices (especially in the VARIOT dataset see table B.3). This precision could maybe be improved with whitelists.

The recall measures the proportion of malicious captures that were flagged as malicious. We saw that the misclassified malicious captures mostly corresponded to C&C. This type of traffic is hard to detect since it is similar to normal traffic, and cannot be detected with our method if the endpoint is not blacklisted or detected by our DGA classifier.

10.2. Evaluation of the resource usage

10.2.1. CPU and RAM usage

Our implementation uses a multi-threading approach. This is necessary to be able to perform continuous ARP spoofing while analyzing the intercepted packets. However, we did not use a multi-core approach. This is first to ease the implementation as threads on the same core can easily share some data. A second motivation for a single-core implementation is to ensure compatibility. Requiring multiple cores could have caused incompatibilities if the program is run on a single-core CPU. Also, as we

Label	Scenario	Type	packets	Kept alerts	Ratio	Classification
1	CTU-Malware 7	C&C, H scan	≈2500000	2505201	1.002	1
1	CTU-Malware 1	C&C, H scan	1686291	2077	0.001	0
1	CTU-Malware 17	C&C, DDoS, H scan	≈2500000	9882507	0.393	1
1	CTU-Malware 20	C&C	50156	0	0.000	0
1	CTU-Malware 21	C&C	50277	0	0.000	0
1	CTU-Malware 3	C&C, H scan	496959	72362	0.146	1
1	CTU-Malware 33	C&C, H scan	≈2500000	1963656	0.785	1
1	CTU-Malware 34	C&C, DDoS	233865	120546	0.515	1
1	CTU-Malware 35	C&C, H scan	≈2500000	708317	0.283	1
1	CTU-Malware 36	H scan	≈2500000	2499346	1.000	1
1	CTU-Malware 39	C&C, H scan	≈2500000	2518641	1.007	1
1	CTU-Malware 42	C&C	24485	0	0.000	0
1	CTU-Malware 43	C&C, H scan	≈2500000	2537578	1.015	1
1	CTU-Malware 44	C&C	1309350	1325498	1.012	1
1	CTU-Malware 48	C&C, H scan	≈2500000	353207	0.141	1
1	CTU-Malware 49	C&C, H scan	≈2500000	842193	0.337	1
1	CTU-Malware 52	C&C, H scan	≈2500000	1267307	0.507	1
1	CTU-Malware 60	C&C, DDoS	≈2500000	2541029	1.016	1
1	CTU-Malware 8	C&C	23623	0	0.000	0
1	CTU-Malware 9	H scan	6437837	6411509	0.996	1
0	CTU-Honeypot 4	benign	8077	0	0.000	0
0	CTU-Honeypot 4-1	benign	21664	0	0.000	0
0	CTU-Honeypot 5	benign	397245	0	0.000	0
0	CTU-Honeypot 7-1	benign	6802	0	0.000	0
0	CTU-Honeypot 7-2	benign	119638	0	0.000	0
0	CTU-Honeypot 7-3	benign	62851	0	0.000	0
0	CTU-Honeypot 7-4	benign	121554	0	0.000	0
0	CTU-Honeypot 7-5	benign	121566	0	0.000	0
0	CTU-Honeypot 7-6	benign	119158	0	0.000	0

Table 10.1: Classification of scenarios on IoT-23 dataset

Figure 10.1: CPU and RAM usage during a run of our software

		Prediction	
		0	1
Label	0	57	6
	1	5	15

Table 10.2: Confusion matrix over the evaluation data

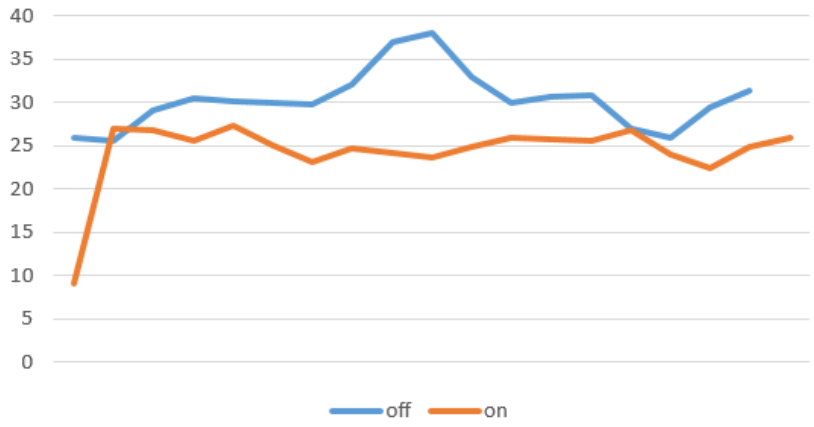


Figure 10.2: Network speed tests with and without interception

only use a single core, we guarantee that multi-core devices remain usable when the tool is running. As shown in figure 10.1, only one CPU is used at 100%, and the others remain unused. Therefore, it is possible to continue to use the computer while the tool runs, which guarantees that our solution does not require a dedicated machine.

The RAM usage is also limited as shown in figure 10.1. The tool only takes about 200Mo of RAM on startup, and does not increase that usage too much over time. We however warn that, as data about flows have to be stored to perform the analysis of network activities, the memory usage may increase significantly if a lot of packets are intercepted.

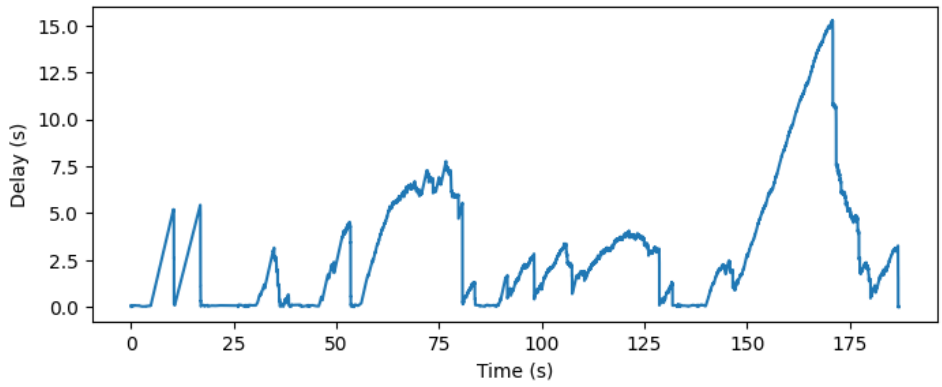
10.2.2. Traffic speed impact

We could not test the influence of NURSE over the network speeds in a controlled environment. However, we performed some experiments in a home network to check the impact of NURSE monitoring on the network speed. Our experiment setup was an Android device which connected to a network speed testing website. We first performed a test without the tool running, then started the tool, monitored the android device and performed a speed test again to measure the difference.

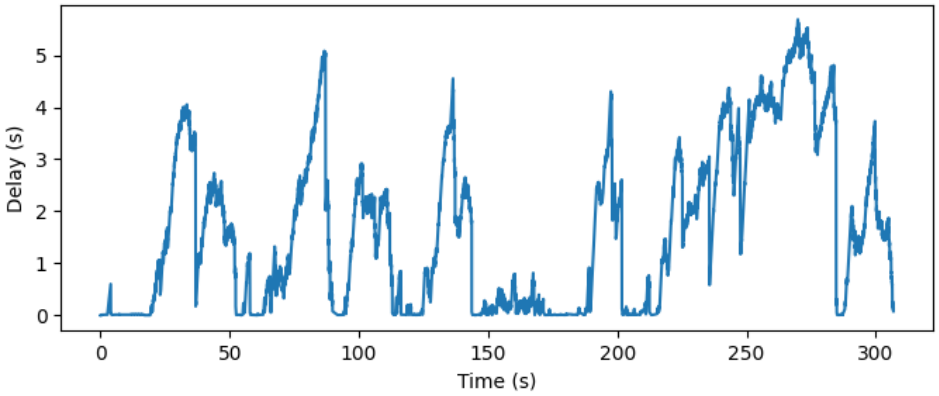
In our home environment, the average speed was 30.46Mbps without the tool running, the graph of the test is shown on figure 10.2. When the tool ran, we observed that it started with a very slow connection of less than 1Mbps, but after a few seconds, the connections went back up to normal speeds. This can be observed in figure 10.2 where the first dot is at about 10Mpbs, before joining back to normal values.

We investigated this issue and discovered that it is caused by the buffering of our packet processing. As we capture packets, the processing of the packet data may take longer than the time before the next packet. Therefore packets are buffered and we can notice a delay between the time the packet is sent and the time it has been processed. The delay is mostly caused by analysis on new domains or IP addresses which require additional checks (scoring with domain classifier, blacklist check...). We notice that, in periods of high activity, the delay may accumulate and exceed 10 seconds. Although this does not considerably affect the botnet detection (since packets are still processed, only with a delay), it can cause some requests to time out, and cause the device to believe that it is offline. This packet delay however can reduce itself after a while as shown in figures 10.3. We note that the delays peaks can reach higher values in Windows, possibly due to the packet capture method on Windows, which relies on an additional dependency.

Since the delay is mostly caused by processing the new domains and new IP addresses, it shrinks



(a) Delay when running on Windows



(b) Delay when running on Linux

Figure 10.3: Packet delays in two web browsing scenarios

to normal values once the checks on new endpoints are performed. That is why, when performing network speed tests, our software always starts with really slow speeds, and reaches higher speeds after a while. Figure 10.2 shows this behavior, with the first point of the curve with interception being lower than the rest. We also noted that some speed tests noted very high latency values that were due to this delay.

With real web-browsing under monitoring, we observe that websites take longer to load especially for images when these are hosted on dedicated subdomains or external domains which trigger new checks for each domain. If the number of new domains or CDNs is too high and cause the delay to exceed the timeout value, this may cause an error on the browser, saying that the page is not available. However, when trying again after a few seconds, as the new domains have been processed, the page loads. The page loading may take longer than without monitoring due to the higher latency caused by the delay of our software.

We obviously consider this slow down and the potential timeout errors as a serious limitations of NURSE since it affects the usability of devices when monitored. However, the speed loss and connection issues are temporary and we note that even if the connection is slow, it still works and our Android device did not report any loss of connection. Therefore, while it could affect some devices that rely on fast communications, we believe that IoT devices that only perform regular exchanges with servers will not be affected. We however do not recommend using NURSE on devices were the network availability or minimal network speeds are critical.

We note that the ARP spoofing and traffic interception does not affect the network speeds of the computer it runs on. However, some functionalities such as the blacklist checks of IP addresses could affect the connection by sending one request per contacted IP. That is why we offer the possibility to disable these functionalities in the configuration page.

10.3. Limitations of the tool

10.3.1. False positives

As mentioned in the previous section, our detection method may raise some false positive alerts. We proposed a way to reduce the number of false positives by whitelisting domains, however, this does not completely solves the false positive issue. Also, even if some domains can be whitelisted in advance to limit false positives, further improvements may also be reactions to false positives instead of being a proactive prevention of false positives.

10.3.2. Privacy considerations

Contrary to IoT-inspector [35], which collects anonymized information on the user side, but analyzes it and displays the dashboard from their servers, NURSE collects, stores, analyzes and displays the information locally. Also no information is stored once the tool is closed. IoT-inspector was built with data collection in mind as its goal is to build an IoT traffic dataset, which justifies that data is sent to a server. However, NURSE is made to run in a local network and only aims at analyzing the traffic in this network. Therefore no data is exported and all the traffic data stays on the device the tool is running on. There is also no telemetry or similar features. Users who are concerned about their traffic information being leaked or sent to external servers can use NURSE knowing that the traffic features stay offline.

However, NURSE cannot run fully offline as it relies on some external services to identify the external IP, check if IP addresses are blacklisted, ease data plotting with JavaScript libraries... These services use free and anonymous APIs, and no identifiable information is sent to these APIs. We also provide the possibility to disable some online functionalities such as blacklist checks if some users do not want to use this API.

A second privacy concern is about people living in the same household, and sharing the same network. A tool that performs traffic interception could easily be misused as a spying tool.

The designers of IoT-inspector had already voiced concerns about their tool being used as a spying tool to observe traffic of devices owned by other people in the same house, which would cause privacy issues. To limit this misuse of their tool, they had installed a small security check, asking the user to provide the MAC address of a device to monitor it. While knowing the MAC address of a device does not prove the ownership of the device, they considered that knowing the MAC address was enough to perform ARP spoofing, which is enough to build a similar tool to IoT-inspector for spying on users. Therefore, their goal was simply to prevent the use of their tool as a spying tool by people who would not be able to perform network interception without it.

We did not implement such a security in NURSE because it would also deter many users from using it since some users may not know how to find the MAC address on their IoT device. However, NURSE is still in a design phase and such security features could be added before a public release. We also note that the use of a Virtual Private Network (VPN) on the monitored device would hide contacted domains and IP addresses from our tool, thus seriously reducing the possibility of spying. Therefore, someone could hide their traffic from NURSE by using a VPN on their devices.

10.4. Incentives for users

As mentioned in section 2.3, IoT-inspector has been downloaded by 10559 users and run for data collection by 5404. This raises the issue of user incentives: how to motivate users to download NURSE?

Of course, the idealistic answer could be that users would download it to prevent botnet attacks. However, botnet attacks tend to target companies and do not affect end-users, except by disrupting the availability of services. Therefore, we look at reasons that could motivate users to download NURSE by finding a personal gain in this operation.

The first reason that could motivate users to download NURSE could be to detect infected devices, which is its main purpose. Users with infected devices tend to be unaware that their devices are infected [68], therefore, our software could help them in knowing that their device is infected. This could be a motivation as users could also be fearful of knowing that a hacker can access their devices, since the devices may also store personal information.

Another possibility could be that this tool would be used by an ISP to clean its network from botnets. The ISP could for example bundle this tool into a diagnostics tool which would allow them to detect whether their customer has infected devices in their home. The ISP would then have access to more precise information in the diagnostics, and could offer more specific assistance to help the users in

removing malware from the infected device. Therefore, such a tool could serve the ISP trying to clean its network from infected hosts, and the ISP clients who would receive better assistance.

Finally, we added some functionalities in NURSE that could motivate users to download it. In addition to the botnet detection, the tool provides visualization of traffic and a domain blocker (see section 4.3) that could motivate users who want to observe the traffic of their devices. This approach is similar to the one of IoT-inspector creators who added network visualizations for users as an incentive to download their tool. We however added the domain blocking feature that was not in IoT inspector and is made possible by our own implementation of ARP spoofing and IP forwarding.

11

Conclusion

11.1. Our contribution

In this research project, we studied the possibility of detecting infected devices in home networks. This first lead us to study existing botnet detection methods, first to understand how they work, and then to list the requirements to use such techniques in a home network. We then developed NURSE: a botnet detection software that works in this environment. This solution performs traffic interception, analysis and detects malicious behavior to flag devices that perform botnet activities. We also worked on the user interface and the installation process in an attempt to make the tool easy to set up, to simplify the setup and to not require knowledge about the network parameters to run the tool.

The solution we designed uses traffic interception techniques to allow our botnet detection module to scan traffic without requiring any network setup from the user and in a transparent way for scanned devices. We then designed a traffic inspection software that detects botnet traffic using rule-based detection. The rules that are used were designed observing actual malware and normal traffic from public datasets and malware traffic captures from a sandbox.

We also extended our solution with a classifier of domains to help in identifying domains generated by domain generation algorithms used in malwares. This classifier combined techniques from state-of-the-art DGA classifiers, but adapted the classification to work on a local machine and run with no need of external databases that would not be available on a home network.

As we proposed a whitelist functionality in NURSE, to allow users to list domains that should not raise alerts, we also studied methods to identify domains that are used by IoT devices. These methods could also be used in a more extensive census of IoT related domains in further research.

We finally evaluated NURSE, focusing on two aspects: the botnet detection task, and the usability of NURSE in home networks. We evaluated our botnet detection method using a public dataset that contained botnet traffic and we also tested our tool on multiple captures of IoT traffic to measure if some devices are less compatible with our approach than others, causing false alerts or using traffic that cannot be analyzed using our method. We also evaluated our tool noting the impact it has on the user and their home network and accounting the limitations that could deter some users from using it.

Therefore, our contribution was to design a botnet detection tool meant to be used in a smart home. As no solution was designed for home networks, we studied the limitations of current techniques and developed NURSE by gathering botnet detection and networking techniques and combining them in order to build a tool that would require no setup or specific hardware and therefore be easy to install and use in a home network. This serves the end users who have no botnet detection technique available they they can use to detect infected devices in their homes. The software could also be useful for ISPs which could use it to better understand their customers' networks and provide more precise help when asking them to clean infected devices in their homes.

11.2. Answers to research questions

In this section, we detail a short answer for each of the research questions we had listed to drive our research project.

1. - How to detect IoT devices that are part of botnets?
    - *What are common botnet detection techniques?*: There exist many botnet detection techniques which we listed in the state of the art in chapter 2. Most of them rely on knowledge about previous attack pattern or about normal traffic to detect suspicious activities.
    - *What do they need to be applied?*: Botnet detection techniques use traffic inspection to detect suspicious networking activity. Is is therefore necessary to mirror or redirect traffic towards the botnet detection system. Some may also require a regular update of external data, such as the detection rules for IDS.
    - *Why are they not used in homes?*: The fact that traffic has to be mirrored requires a controlled network environment. Botnet detection techniques are designed with this assumption, but most users are not able to set up traffic mirroring in their home.
2. Can detection be performed in a home network?
    - *What are the requirements for a home based botnet detector?*: The requirements we found when designing our solution were: that we have to perform the traffic interception, that we cannot rely on external data such as passive DNS (for space or bandwidth limitations) and that we have to design a tool that is easy to install on Windows, the most used operating system
    - *What can be captured and used as input for the detection?*: Using ARP spoofing, we managed to set up a MITM position. We therefore can use all the intercepted traffic as input for detection. We note that the packets contents remain encrypted when they are. More advanced MITM attacks could allow decryption, but would require some additional setup or more control over the scanned device, which may not always be possible.
    - *How can this be used for detection? Can this be used to apply botnet detection techniques?*: Since we can parse packets headers, we keep track of the opened flows and set up rules to detect suspicious behavior such as DDoS or port scan in these flows. We also take advantage of DNS which is not encrypted to analyze the contacted domains.
    - *How effective are the botnet detection techniques in this home network?*: Detection was effective on the testing dataset, with port scanning and DDoS activities being correctly flagged. However, command and control traffic which looks too much as normal traffic, was missed. The detection also raised some false alerts on normal traffic.
3. Can it work in any home?
    - *What are the supported platforms?*: We developed NURSE using Python. This language is compatible with most operating systems, but requires a python environment to be installed which is not always the case. If the user has no python environment, we also provide an executable version for Windows that does not require Python to run.
    - *Can the installation be made simple, requiring no setup?*: The tool depends on scapy which works natively on Linux and Mac. On Windows, an additional software is required, we wrote a small script to assist the user in installing this script.
    - *How much permissions are needed?*: Scapy require administrator permissions on Linux. On Windows, such permissions are not needed on each run, but installing the required dependencies requires the installer to be an administrator of the system.
    - *How much does this affect the scanned devices?*: NURSE uses ARP spoofing to intercept traffic which is transparent and therefore, would not trigger alerts or abnormal behavior on the scanned device. We however noticed that the traffic interception can cause the network speed to drop when many packets are sent. While this should not affect the device too much, it may cause issues for systems in which the connectivity is crucial.

11.3. Results discussion

In this research project, we designed a botnet detection method that can be used in a home network. We here detail its advantages and limitations.

11.3.1. Advantages

In our solution, we developed a botnet detection method that does not require networking setup to work. The devices are identified mostly passively by listening to broadcast traffic, and sometimes with active queries to obtain their advertised device name. The traffic interception is then done automatically using ARP spoofing, once the user selects the device to monitor. Finally, the use of ARP spoofing makes it possible to perform such an interception with no additional hardware which makes our solution easier to use than other network tools that require dedicated machines. The use of ARP spoofing also allows us to perform traffic interception transparently which is important for a botnet detection tool. Otherwise, the malware could attempt to detect when the detection tools are run and stop its malicious activities during scans.

Our botnet detection solution uses rules to detect malicious behavior. This implementation is simple and does not rely on complex computations which would take too long which allows us to perform live detection. We evaluated it on captures of malicious traffic and noted that the targeted malicious behaviors were detected.

We also took advantage of the MITM we set to perform botnet detection and of the traffic features we gather to propose additional features in NURSE. We exploited the MITM position and our new implementation of IP forwarding to propose a domain blocker which could be used to prevent malicious domains from being contacted or to block some domains that the user do not want to send information to, such as tracking domains. We also proposed some visualizations of the intercepted traffic. While we do not consider these to be part of the detection method, they could help in going deeper into the traffic to determine whether some devices are behaving suspiciously, without having to manually inspect the packets.

11.3.2. Limitations

In the evaluation, we noted that our botnet detection solution was not perfect. It could detect the malicious behaviors in the test datasets, which is the intended behavior, but we noted that it also raised some alerts on normal traffic. Therefore, it could cause some users to think that their device is infected when it is not. However, we could limit the impact of false positives by helping users when an infection alert is shown. This help could for example be provided via help pages in the webserver of NURSE, in which we would write tutorials about what to do when alerts are shown. Another possibility is that the support of an ISP company which uses our software could investigate the alerts and help users in determining whether this were false or not, before helping the users in cleaning their devices in case of an infection.

A drawback of our implementation is that it slows down the network connection of the monitored devices. Of course, the MITM position we set makes it hard not to affect the connection, however the slow down could be caused by the analysis time. Therefore, a more optimized implementation may manage to limit the effect it has on connections. This is not a major limitation for a home usage (even if it may cause some users to be reluctant to use it), but may cause NURSE not to be usable to analyze the behavior of devices for which the connectivity is crucial. While the impact of a connection loss for a connected oven may be limited, especially if the oven is not in use, it could be more serious for health-monitoring devices.

The goal or our research was to design a botnet detection method which can be used at home, which we did. But the solution we designed is not ready to be used in users homes and is more a proof-of-concept tool. Unfortunately, we could not perform real tests, for example to detect real malwares in controlled environment, or to test the user reception to the tool by having it used by real users. We also could not perform full software testing and therefore cannot guarantee that the software will run flawlessly on the supported operating systems, especially as it performs ARP spoofing which is an advanced use of the network card of the device. Therefore, even if we designed the solution and shows that botnet detection can be made accessible to home users, NURSE is not completely ready to be used by home users.

As with the extensive testing of NURSE, some features are still incomplete. The main example is the whitelist of domains. The goal of this feature is to prevent false positives by listing safe domains. However, it will only be effective with a extensive list. We could grab some IoT related domains that we considered safe from IoT traffic captures and using the methods we described in chapter 8. But the list is still incomplete, and therefore false positives may still occur. Also, the idea of listing the domains is quite limited since the IoT domains list may evolve over time, causing our whitelist to be less efficient

as time passes.

11.4. Future work

Our work consisted of developing a proof of concept software performing botnet detection for users at home. We listed its limitations, and now present ways in which we think that our research could be expanded in multiple ways:

11.4.1. Improving the botnet detection technique

Our software could be improved first by working on the botnet detection method. In our method we used a rule based approach to detect malicious software. However, our traffic analysis module could be replaced with another, for example to implement a different botnet detection method. One could then evaluate whether different botnet detection methods perform better in home networks. Other botnet detection methods could also be compared with our solution looking at whether they fix some of the limitations of our implementation, for example in terms of resource usage or network speed.

These improvements could also be done by working on optimization. NURSE is designed in Python. We chose Python because we were familiar with this language and its packet analysis capacities. One could however work on re-implementing our solution in another language. The implementation could be done with a focus on:

- Optimization: we noted that our tool caused slowdowns on the network speed, possibly due to the time it takes to process packets. One could therefore focus on optimizing our implementation to make it faster.
- Security: we noted that our tool requires high permissions to perform its packet interception. Running a script with high permissions could cause security vulnerabilities with serious impact.
- Stability: an ISP interested in using our solution for its customers could also perform advanced tests to prevent crashes. Another solution could be to re-implement it in a language that is more stable to ensure compatibility and stability in operating systems. Rust, for example, offers more controls over the data ownership which could prevent race conditions in our multi-threaded tool.

11.4.2. Propose additional features

As we focused on designing a solution to perform botnet detection at home, NURSE contains little additional features. However, more features could be added for the users, for example: we showed how DNS spoofing could be added to perform domain blocking. Visualizations could also be added to help users in analyzing the traffic or suspicious events that have been detected.

Additional features could also be added for ISPs who want their customers to use this tool. We already propose a functionality to generate a report that contains some information about identified devices, intercepted traffic and raised alerts... However, ISPs may want to use this tool to list the devices their customers own, and to provide more precise help for malware cleaning. This could be achieved by adding more features to our tool.

The code of NURSE is available in open-source (link) therefore letting anyone download it and add functionalities.

11.4.3. Test NURSE in real conditions

Test the botnet detection

To evaluate our botnet detection method, we tested it on public datasets containing traffic captures of normal and infected IoT devices. The behavior of our tool with these datasets as input should not differ from its behavior with live packet capture. However, we could not assess this as we did not have a safe environment with IoT devices that could be infected with malware (while ensuring that the malware would not spread or cause uncontrolled damages). It could be interesting to check how the detection would work with devices that send normal traffic and malicious traffic at the same time, or if the software manages to detect devices in the network in normal conditions for example with devices in idle state.

Test the user reception

We designed NURSE for end-users, hoping that it would make botnet detection accessible to users who may not know how to setup previously existing botnet detection in their home network.

However, we could not ask users to test NURSE to understand how they receive it. Before recommending it to their customers, Internet Service Providers may want to answer the following questions:

- Can all users understand how to install the software and run it?
- Can all users understand the detection results?
- How do users react if a device is flagged as infected?

Previous research has been conducted on studying the reception of ISP notifications for infected IoT devices [5]. Therefore, further research could attempt to measure if a software such as ours improves the way users detect and clean infected IoT devices.

11.4.4. The IoT domain census

In chapter 8, we studied ways to detect domains that are likely to be dedicated to IoT devices. Listing these domains extensively could help our botnet detection by providing a list of domains that can be considered as normal traffic and therefore should not be flagged, thus reducing the risk of false alerts on normal traffic.

Such a census of IoT domains would also be a major contribution for IoT identification research. As shown in section 2.2.2, most current identification techniques rely on domain names. A method to list domains used by IoT devices without having to own the models would therefore help IoT identification research. The methods we presented were only interpretations of visualizations that can reveal some IoT dedicated domains starting from a domain found in a capture and using passive DNS data. However, we hope that future work could lead to automated detection techniques which could identify IoT dedicated domains or distinguish them from human-facing domains.

Bibliography

[1] Johannes Bader (baderj). *domain generation algorithms*. Apr. 2021. URL: https://github.com/baderj/domain_generation_algorithms.

[2] Sharad Agarwal, Pascal Oser, and Stefan Lueders. "Detecting IoT Devices and How They Put Large Heterogeneous Networks at Security Risk". In: *Sensors* 19.19 (2019). ISSN: 1424-8220. DOI: 10.3390/s19194107. URL: https://www.mdpi.com/1424-8220/19/19/4107.

[3] Alexa. *Alexa Top Sites for Countries*. URL: https://www.alexa.com/topsites/countries.

[4] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. "Sok: Security evaluation of home-based iot deployments". In: *2019 IEEE symposium on security and privacy (sp)*. IEEE. 2019, pp. 1362–1380.

[5] Lisette Altena. "Exploring effective notification mechanisms for infected IoT devices". In: *TU Delft Technology, Policy and Management* (2018).

[6] Amazon. *AWS IoT Platform*. 2018. URL: https://aws.amazon.com/iot/.

[7] andrewaeva. *DGA*. July 2017. URL: https://github.com/andrewaeva/DGA.

[8] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. "Building a dynamic reputation system for dns." In: *USENIX security symposium*. 2010, pp. 273–290.

[9] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. "From throw-away traffic to bots: Detecting the rise of DGA-based malware". In: *21st {USENIX} Security Symposium ({USENIX} Security 12)*. 2012, pp. 491–506.

[10] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. "Keeping the smart home private with smart (er) IoT traffic shaping". In: *Proceedings on Privacy Enhancing Technologies* 2019.3 (2019), pp. 128–148.

[11] Kevin Ashton. *That 'Internet of Things' Thing*. June 2009. URL: https://www.rfidjournal.com/that-internet-of-things-thing.

[12] John Bambenek. *DGA feed*. URL: https://osint.bambenekconsulting.com/feeds/dga-feed.txt.

[13] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis." In: Jan. 2011.

[14] Philippe Biondi and the Scapy community. *Scapy*. URL: https://scapy.net/.

[15] Caida. *State of IP spoofing*. May 2021. URL: https://spoofer.caida.org/summary.php.

[16] chrislee35. *Rubot, Botnet Emulation Framework*. Dec. 2015. URL: https://github.com/chrislee35/rubot/blob/master/lib/rubot/control/dga/bamital.rb.

[17] Plummer D. *An Ethernet Address Resolution Protocol*. RFC 826. RFC Editor, Nov. 1982. URL: https://datatracker.ietf.org/doc/html/rfc826.

[18] Pedro Marques Da Luz. "Botnet detection using passive DNS". In: *Radboud University: Nijmegen, The Netherlands* (2014).

[19] Rohan Doshi, Noah Apthorpe, and Nick Feamster. "Machine learning ddos detection for consumer internet of things devices". In: *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2018, pp. 29–35.

[20] Chris Duckett. *Chromium DNS hijacking detection accused of being around half of all root queries*. Aug. 2020. URL: https://www.zdnet.com/article/chromium-dns-hijacking-detection-accused-of-being-around-half-of-all-root-queries/.

[21] Owen P Dwyer, Angelos K Marnerides, Vasileios Giotsas, and Troy Mursch. "Profiling IoT-based Botnet Traffic using DNS". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2019, pp. 1–6.

[22] ENISA. *Baseline Security Recommendations for IoT in the context of Critical Information Infrastructures*. 2017. URL: https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot.

[23] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. “A Survey of Botnet and Botnet Detection”. In: *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. 2009, pp. 268–273. DOI: 10.1109/SECURWARE.2009.48.

[24] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. “Measuring HTTPS Adoption on the Web”. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1323–1338. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/felt.

[25] Sean Gallagher. *Reddit-powered botnet infected thousands of Macs worldwide*. Apr. 2014. URL: https://arstechnica.com/information-technology/2014/10/reddit-powered-botnet-infected-thousands-of-macs-worldwide/.

[26] S. García, M. Grill, J. Stiborek, and A. Zunino. “An empirical comparison of botnet detection methods”. In: *Computers & Security* 45 (2014), pp. 100–123. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2014.05.011.

[27] Google. *Google SafeBrowsing API*. URL: https://developers.google.com/safe-browsing/v4/.

[28] Google. *HTTPS encryption on the web*. 2020. URL: https://transparencyreport.google.com/https/overview.

[29] Guofei Gu, Junjie Zhang, and Wenke Lee. “BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic”. In: (Jan. 2008).

[30] Hang Guo and John Heidemann. “Detecting IoT Devices in the Internet”. In: *IEEE/ACM Transactions on Networking* 28.5 (2020), pp. 2323–2336. DOI: 10.1109/TNET.2020.3009425.

[31] hanzhang0116. *BotDigger*. Feb. 2018. URL: https://github.com/hanzhang0116/BotDigger.

[32] Eszter Hargittai, Anne Marie Piper, and Meredith Ringel Morris. “From internet access to internet skills: digital inequality among older adults”. In: *Universal Access in the Information Society* 18.4 (2019), pp. 881–890.

[33] hmaccelerate. *DGA_Detection*. Sept. 2019. URL: https://github.com/hmaccelerate/DGA_Detection.

[34] Pi-hole. *Pi-hole*. 2021. URL: https://pi-hole.net/.

[35] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. “IoT Inspector”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4.2 (June 2020), pp. 1–21. ISSN: 2474-9567. DOI: 10.1145/3397333. URL: http://dx.doi.org/10.1145/3397333.

[36] Marcus Hutchins. *How to Accidentally Stop a Global Cyber Attacks*. May 2017. URL: https://www.malwaretech.com/2017/05/how-to-accidentally-stop-a-global-cyber-attacks.html.

[37] *Internet Census 2012*. URL: http://census2012.sourceforge.net/paper.html.

[38] M. O’Reirdan J. Livingood N. Mody. *An Ethernet Address Resolution Protocol*. RFC 6561. RFC Editor, Mar. 2012. URL: https://datatracker.ietf.org/doc/html/rfc6561.

[39] JavaTpoint. *Man-in-the-middle (MITM) Attacks*. 2018. URL: https://www.javatpoint.com/cyber-security-mitm-attacks.

[40] jgamblin. *scanner.c: mirai source code*. Oct. 2016. URL: https://github.com/jgamblin/Mirai-Source-Code/blob/master/mirai/bot/scanner.c.

[41] Te-k. *Pysafebrowsing*. URL: https://pypi.org/project/pysafebrowsing/.

[42] keredson. *Wordninja*. Aug. 2019. URL: https://github.com/keredson/wordninja.

[43] Eduard Kovacs. *150,000 IoT Devices Abused for Massive DDoS Attacks on OVH*. Sept. 2016. URL: https://www.securityweek.com/150000-iot-devices-abused-massive-ddos-attacks-ovh.

[44] Bharath Kumar. *The Art of packet crafing with Scapy, ARP spoofing*. 2020. URL: https://0xbharath.github.io/art-of-packet-crafting-with-scapy/network_attacks/arp_spoofing/index.html.

[45] Knud Lasse Lueth. *State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time*. Nov. 2020. URL: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[46] Majestic. *Majestic million domain list*. URL: https://fr.majestic.com/reports/majestic-million.

[47] Linda Markowsky and George Markowsky. "Scanning for Vulnerable Devices in the Internet of Things". In: Oct. 2015. DOI: 10.1109/IDAACS.2015.7340779.

[48] mplx. *mac2vendor*. 2021. URL: https://mac2vendor.com/.

[49] Phil Muncaster. *Google Reveals it Was Hit by 2.5Tbps DDoS*. Oct. 2020. URL: https://www.infosecurity-magazine.com/news/google-reveals-it-was-hit-by/.

[50] Netlab. *Netlab OpenData Project*. Mar. 2021. URL: https://data.netlab.360.com/dga/.

[51] NetMarketShare. *Operating System Market Share*. Oct. 2020. URL: https://netmarketshare.com/operating-system-market-share.aspx.

[52] OpenDNS. *public domain lists*. Nov. 2014. URL: https://github.com/opendns/public-domain-lists.

[53] Mockapetris P. *Domain names - Implementation and Specification*. RFC 1035. RFC Editor, Nov. 1987. URL: https://datatracker.ietf.org/doc/html/rfc1035.

[54] Danny Palmer. *This massive DDoS attack took large sections of a country's internet offline*. May 2021. URL: https://www.zdnet.com/article/this-massive-ddos-attack-took-large-sections-of-a-countrys-internet-offline/.

[55] A Parmisano, S Garcia, and MJ Erquiaga. "A Labeled Dataset with Malicious and Benign IoT Network Traffic". In: *Stratosphere Laboratory: Praha, Czech Republic* (2020).

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[57] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. "IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis". In: *2020 IEEE European Symposium on Security and Privacy (EuroS P)*. 2020, pp. 474–489. DOI: 10.1109/EuroSP48549.2020.00037.

[58] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. "IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis". In: *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2020, pp. 474–489.

[59] PortSwigger. *Burp Suite, proxy setup*. 2021. URL: https://portswigger.net/burp/documentation/desktop/getting-started/proxy-setup/browser.

[60] Marty Roesch and the Snort team. *Snort*. 2021. URL: https://www.snort.org.

[61] Imanol Rubio-Unanue, Iñigo Ortega-Martinez, Markel Baskaran, Eneko Bermejo, Antton Rodriguez, Igor Santos, Iñaki Garitano, and Urko Zurutuza. *IoT security - network traffic under normal conditions*. Mondragon Unibertsitatea: Data Analysis and Cybersecurity Research Group [producer]. European Data Portal [distributor]. Mar. 2021. URL: https://data.europa.eu/data/datasets?keywords=variot.

[62] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J. Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. “A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild”. In: *Proceedings of the ACM Internet Measurement Conference*. IMC ’20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 87–100. ISBN: 9781450381383. DOI: 10.1145/3419394.3423650. URL: https://doi.org/10.1145/3419394.3423650.

[63] Farsight Security. *DNSDB*. 2020. URL: https://docs.dnsdb.info/.

[64] AWS Shield. *Threat Landscape Report (Q1 2020)*. 2020. URL: https://aws-shield-tlr.s3.amazonaws.com/2020-Q1_AWS_Shield_TLR.pdf.

[65] Donald Shin. *How to Defend Against Amplified Reflection DDoS Attacks*. July 2018. URL: https://www.a10networks.com/blog/how-defend-against-amplified-reflection-ddos-attacks/.

[66] *Shodan: the search engine for the internet of things*. URL: https://www.shodan.io/.

[67] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. “Classifying IoT devices in smart environments using network traffic characteristics”. In: *IEEE Transactions on Mobile Computing* 18.8 (2018), pp. 1745–1759.

[68] Anthony Spadafora. *Bots and botnets spread worldwide by unaware users*. 2018. URL: https://www.itproportal.com/news/global-spread-of-bots-and-botnets-spurred-on-by-unaware-users/.

[69] Spamhaus. *The Spamhaus project*. URL: https://www.spamhaus.org/.

[70] Spartan2194. *Detecting SSH brute forcing with Zeek*. Apr. 2019. URL: https://holdmybeersecurity.com/2019/04/17/detecting-ssh-brute-forcing-with-zeek/.

[71] *sshd_config(5) Linux man page*. Apr. 2013. URL: https://linux.die.net/man/5/sshd%7B%5C_%7Dconfig.

[72] Zyxel support. *How can the ”tcpdump” command be used to capture packets? How can captured packets be save as Wireshark compatible files in the SBG3300?* 2021. URL: https://kb.zyxel.com/KB/searchArticle!gwsViewDetail.action?articleOid=014926&lang=EN.

[73] CyberNews Team. *We infiltrated an IRC botnet. Here’s what we found*. Nov. 2020. URL: https://cybernews.com/security/we-infiltrated-an-irc-botnet-heres-what-we-found/.

[74] PyInstaller Development Team. *Pyinstaller*. Jan. 2021. URL: https://www.pyinstaller.org/.

[75] the Wireshark team. *Wireshark*. 2021. URL: https://www.wireshark.org/.

[76] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. “IoT security: ongoing challenges and research opportunities”. In: *2014 IEEE 7th international conference on service-oriented computing and applications*. IEEE. 2014, pp. 230–234.

[77] Guodong Zhao, Ke Xu, Lei Xu, and Bo Wu. “Detecting APT malware infections based on malicious DNS and traffic analysis”. In: *IEEE access* 3 (2015), pp. 1132–1142.

A

Additional figures

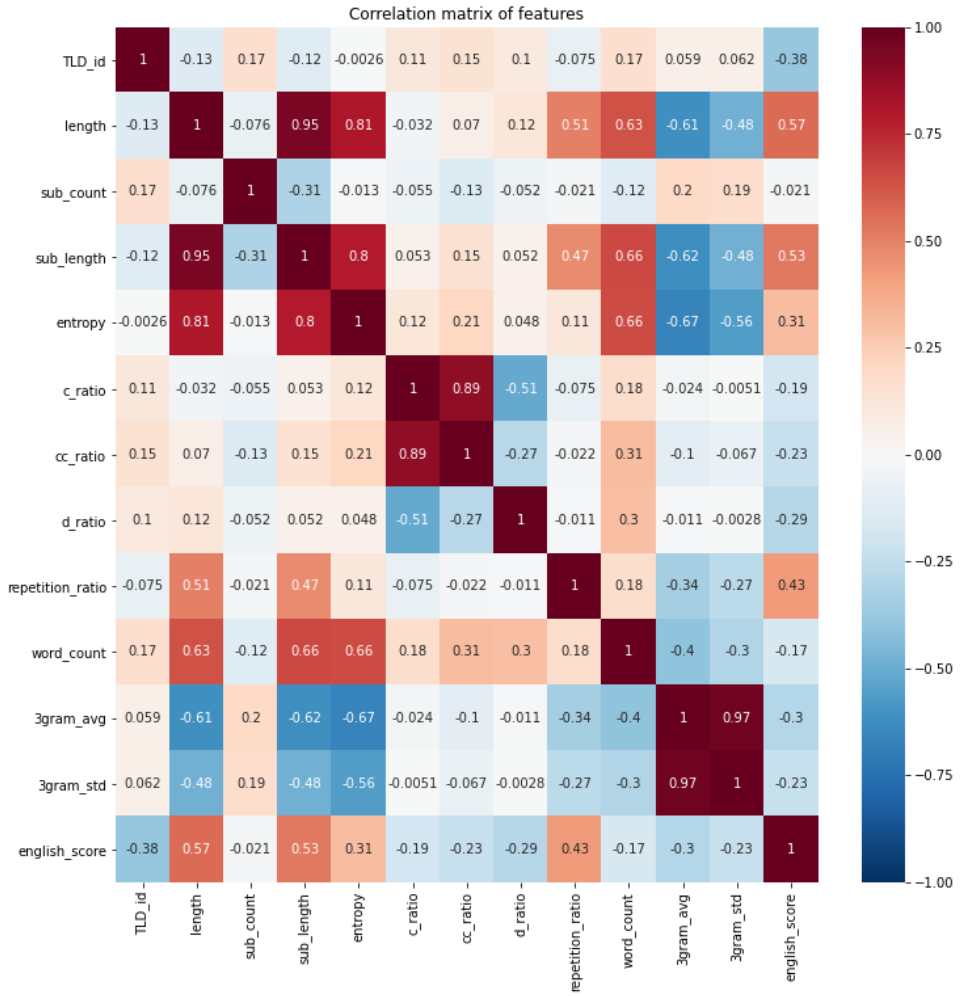


Figure A.1: Correlation matrix of domain features

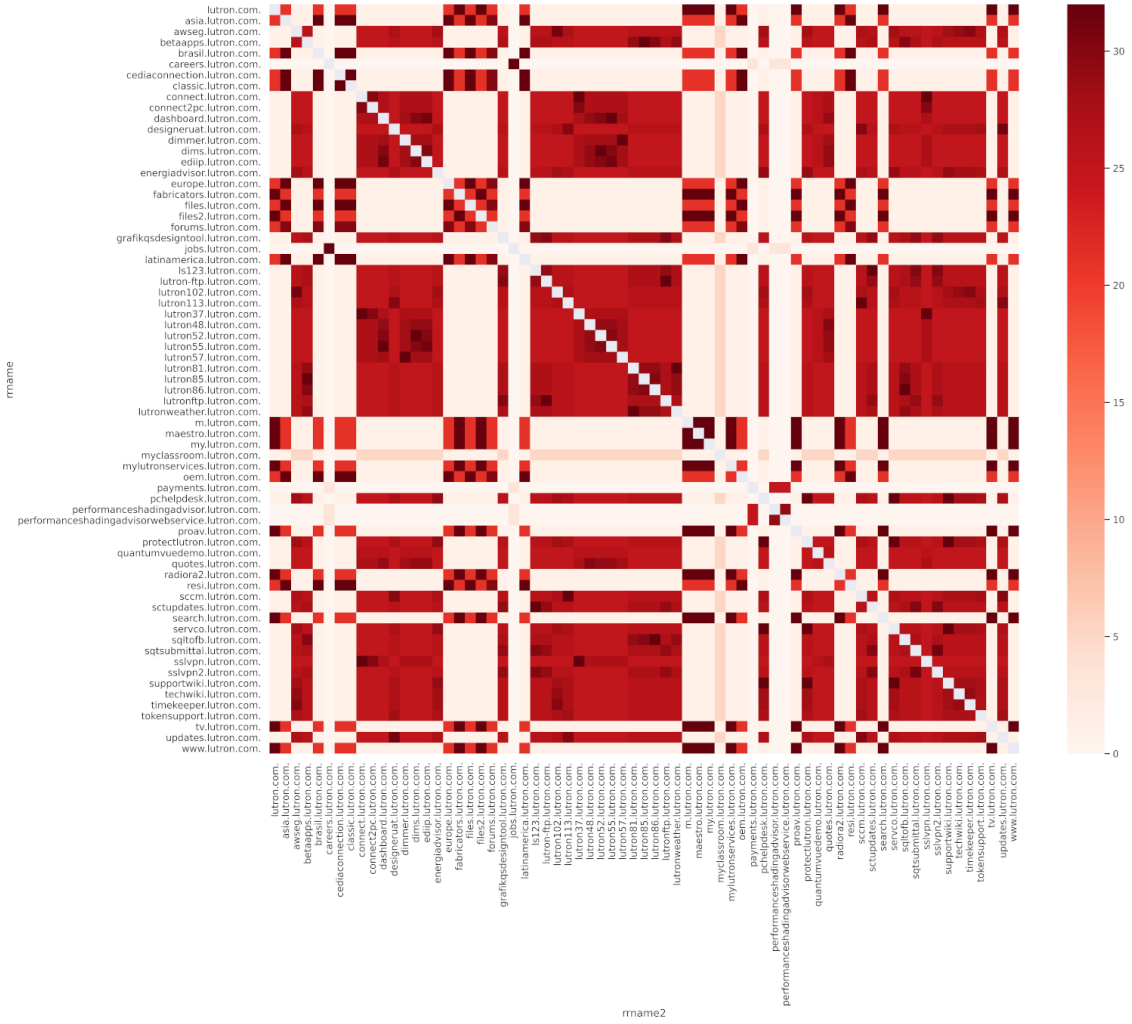


Figure A.2: Distance between Lutron subdomains

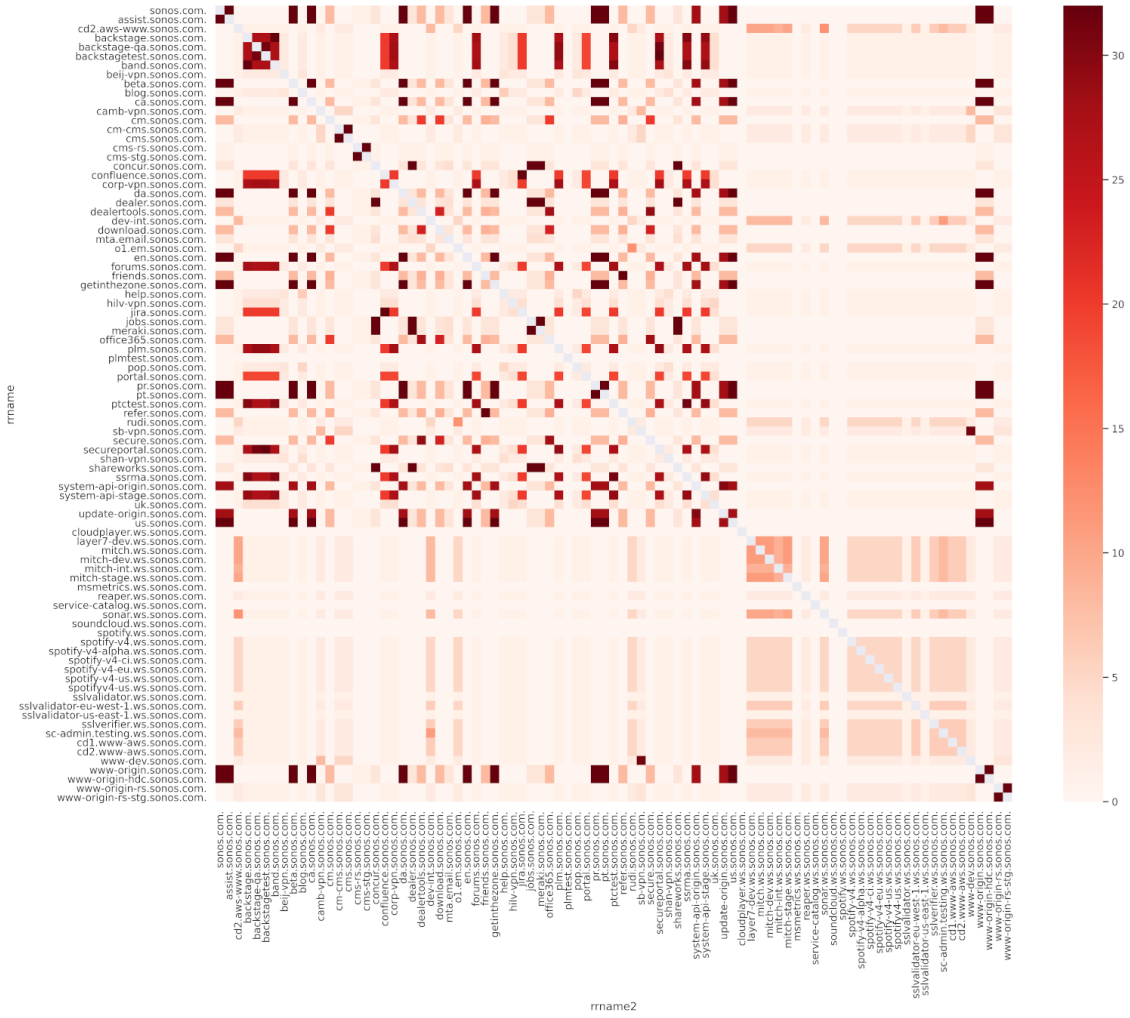


Figure A.3: Distance between Sonos subdomains

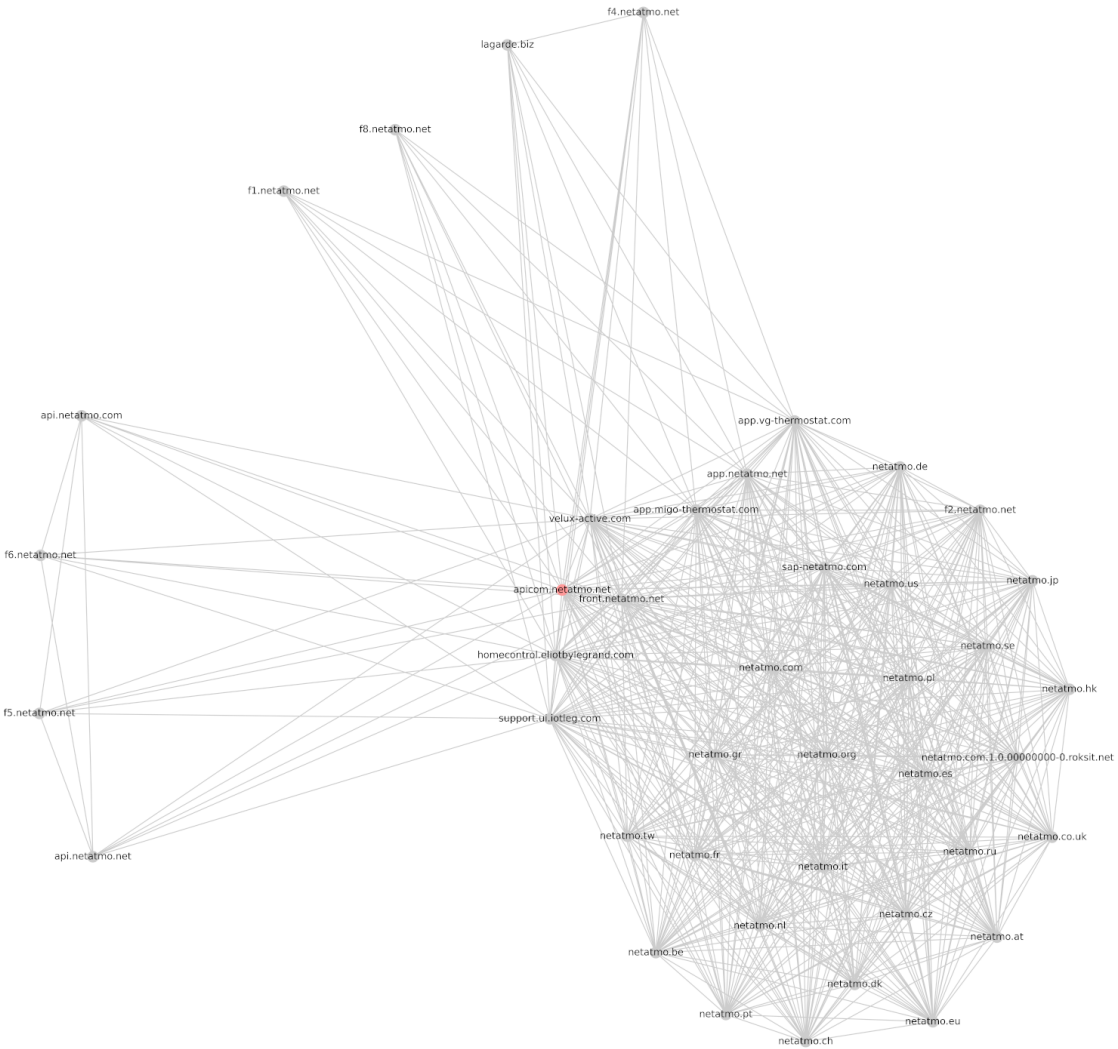


Figure A.4: Graph of netatmo domains

B

Detailed alerts

Scenario	Type	packets	H. port scan	DoS packets	Hardcoded IP
CTU-Malware 7	C&C, H scan	≈2500000	2505201	0	2506411
CTU-Malware 1	C&C, H scan	1686291	377	1700	346590
CTU-Malware 17	C&C, DDoS, H scan	≈2500000	982507	0	1267942
CTU-Malware 20	C&C	50156	0	0	0
CTU-Malware 21	C&C	50277	0	0	0
CTU-Malware 3	C&C, H scan	496959	72362	0	74915
CTU-Malware 33	C&C, H scan	≈2500000	1963656	0	2533130
CTU-Malware 34	C&C, DDoS	233865	0	120546	778
CTU-Malware 35	C&C, H scan	≈2500000	708317	0	708342
CTU-Malware 36	H scan	≈2500000	2499346	0	2499609
CTU-Malware 39	C&C, H scan	≈2500000	2518641	0	2518641
CTU-Malware 42	C&C	24485	0	0	0
CTU-Malware 43	C&C, H scan	≈2500000	2537578	0	2537512
CTU-Malware 44	C&C	1309350	0	1325498	4
CTU-Malware 48	C&C, H scan	≈2500000	353207	0	353207
CTU-Malware 49	C&C, H scan	≈2500000	842193	0	842261
CTU-Malware 52	C&C, H scan	≈2500000	1267307	0	1267306
CTU-Malware 60	C&C, DDoS	≈2500000	1267307	2541029	21
CTU-Malware 8	C&C	23623	0	0	2743
CTU-Malware 9	H scan	6437837	6411363	146	7523827
CTU-Honeypot 4	benign	8077	0	0	1
CTU-Honeypot 4-1	benign	21664	0	0	0
CTU-Honeypot 5	benign	397245	0	0	0
CTU-Honeypot 7-1	benign	6802	0	0	0
CTU-Honeypot 7-2	benign	119638	0	0	0
CTU-Honeypot 7-3	benign	62851	0	0	0
CTU-Honeypot 7-4	benign	121554	0	0	0
CTU-Honeypot 7-5	benign	121566	0	0	0
CTU-Honeypot 7-6	benign	119158	0	0	0

Table B.1: Alerts raised on IoT-23 dataset

Scenario	n_packets	DGA	Spoofed IP	Hardcoded	DoS
amazon-EchoDot3rdGen-normal-2	9848	0	0	0	0
mikrotik-RB951Ui2HnD-normal-2	104	0	10	1	0
mikrotik-RB951Ui2HnD-normal	6004	0	2	1	0
mobvoi-TicWatchE2-normal-2	16122	0	3	2	2462
mobvoi-TicWatchE2-normal	89622	0	11	14	341
netatmo-HealthyHomeCoach-normal	2118	0	0	2	0
netatmo-SmartAlarmSystemWithCamera-normal-2	10470	11	4	1	1730
netatmo-SmartAlarmSystemWithCamera-normal-3	15446	11	0	1	2214
netatmo-SmartAlarmSystemWithCamera-normal	5345	20	4	1	639
netatmo-SmartHomeWeatherStation-normal-2	1149	0	4	1	0
netatmo-SmartHomeWeatherStation-normal-3	2987	0	0	0	0
netatmo-SmartHomeWeatherStation-normal	3193	0	3	0	0
netatmo-SmartSmokeAlarm-normal	24121	0	3	1	234
raspberry-pi4modelb-normal-2	240745	0	1	0	0
raspberry-pi4modelb-normal-3	74	0	3	0	0
raspberry-pi4modelb-normal-4	194930	0	1	0	0
raspberry-pi4modelb-normal	112	0	6	0	0
smarter-Coffee2ndGeneration-normal-2	866	0	0	0	0
smarter-Coffee2ndGeneration-normal	719	0	3	0	0
smarter-FridgeCam-normal	40691	0	12	4	0
smarter-iKettle-normal-2	308	0	2	0	0
smarter-iKettle-normal	3598	0	6	0	0
tplink-hs110-normal-2	1769	0	6	0	0
tplink-hs110-normal	1997	0	5	0	0
amazon-EchoDot3rdGen-normal	18861	0	6	1	0
leotec-AndroidTvBoxGCX2432-normal	92147	0	2	1	0
netatmo-SmartRainGauge-normal	1446	0	7	0	0
tplink-tapoC200-normal-2	3818	0	203	1	0
tplink-tapoC200-normal	2663	0	1	1	0
tplink-tapoL510E-normal-2	838	0	0	1	0
tplink-tapoL510E-normal	2670	0	0	1	0
xiaomi-MiAirPurifier3H-normal	10049	0	1	1	0
xiaomi-MiBoxSGlobalVersion-normal-2	171826	0	2	1	4361
xiaomi-MiBoxSGlobalVersion-normal	720829	0	3	1	146
xiaomi-MiHomeSecurityBasicCamera1080p-normal-2	3267	0	4	1	0
xiaomi-MiHomeSecurityBasicCamera1080p-normal	26707	0	6	5	1265
xiaomi-MiLEDCeilingLight-normal	19902	0	10	3	0
xiaomi-MiLEDSmartBulb-normal	2612	0	0	1	0
xiaomi-MiSmartCompactProjector-normal-2	162895	0	2	1	0
xiaomi-MiSmartCompactProjector-normal	25537	0	3	1	0
xiaomi-MiSmartSensorSet-normal	42769	0	3	0	0
amazon-EchoShow5-normal-2	1367195	0	2	0	0
amazon-EchoShow5-normal	1272456	0	0	0	0
beagleboard-BeagleBoneAI-normal	367659	0	3	0	0
google-ChromecastUltra-normal	131947	0	1	2	282
google-Home-normal	82198	0	1	1	0
google-HomeMini-normal-2	13760	0	12	0	4071
google-HomeMini-normal	83530	0	1	1	2142
google-NestHub-normal-2	24902	0	2	0	2104
google-NestHub-normal	166627	0	1	1	0
google-NestMini-normal-2	22237	0	2	0	6770
google-NestMini-normal	150036	0	1	2	1796
honeywell-C2WiFiHomeSecurityCamera-normal	47437	0	0	1	0
honeywell-W1WaterLeakAndFreezeDetector-normal	196	0	7	0	0
leotec-AndroidTvBoxGCX2432-normal-2	200974	0	2	1	0

Table B.2: Alerts raised on benign traffic in VARIoT dataset

Label	Scenario	Type	packets	Kept alerts	Ratio	Classification
0	amazon-EchoDot3rdGen-normal-2	benign	9848	0	0.000	0
0	mikrotik-RB951Ui2HnD-normal-2	benign	104	0	0.000	0
0	mikrotik-RB951Ui2HnD-normal	benign	6004	0	0.000	0
0	mobvoi-TicWatchE2-normal-2	benign	16122	2462	0.153	1
0	mobvoi-TicWatchE2-normal	benign	89622	341	0.004	0
0	netatmo-HealthyHomeCoach-normal	benign	2118	0	0.000	0
0	netatmo-SmartAlarmSystemWithCamera-normal-2	benign	10470	1741	0.166	1
0	netatmo-SmartAlarmSystemWithCamera-normal-3	benign	15446	2225	0.144	1
0	netatmo-SmartAlarmSystemWithCamera-normal	benign	5345	659	0.123	1
0	netatmo-SmartHomeWeatherStation-normal-2	benign	1149	0	0.000	0
0	netatmo-SmartHomeWeatherStation-normal-3	benign	2987	0	0.000	0
0	netatmo-SmartHomeWeatherStation-normal	benign	3193	0	0.000	0
0	netatmo-SmartSmokeAlarm-normal	benign	24121	234	0.010	0
0	raspberry-pi4modelb-normal-2	benign	240745	0	0.000	0
0	raspberry-pi4modelb-normal-3	benign	74	0	0.000	0
0	raspberry-pi4modelb-normal-4	benign	194930	0	0.000	0
0	raspberry-pi4modelb-normal	benign	112	0	0.000	0
0	smarter-Coffee2ndGeneration-normal-2	benign	866	0	0.000	0
0	smarter-Coffee2ndGeneration-normal	benign	719	0	0.000	0
0	smarter-FridgeCam-normal	benign	40691	0	0.000	0
0	smarter-iKettle-normal-2	benign	308	0	0.000	0
0	smarter-iKettle-normal	benign	3598	0	0.000	0
0	tplink-hs110-normal-2	benign	1769	0	0.000	0
0	tplink-hs110-normal	benign	1997	0	0.000	0
0	amazon-EchoDot3rdGen-normal	benign	18861	0	0.000	0
0	leotec-AndroidTvBoxGCX2432-normal	benign	92147	0	0.000	0
0	netatmo-SmartRainGauge-normal	benign	1446	0	0.000	0
0	tplink-tapoC200-normal-2	benign	3818	0	0.000	0
0	tplink-tapoC200-normal	benign	2663	0	0.000	0
0	tplink-tapoL510E-normal-2	benign	838	0	0.000	0
0	tplink-tapoL510E-normal	benign	2670	0	0.000	0
0	xiaomi-MiAirPurifier3H-normal	benign	10049	0	0.000	0
0	xiaomi-MiBoxSGlobalVersion-normal-2	benign	171826	4361	0.025	0
0	xiaomi-MiBoxSGlobalVersion-normal	benign	720829	146	0.000	0
0	xiaomi-MiHomeSecurityBasicCamera1080p-normal-2	benign	3267	0	0.000	0
0	xiaomi-MiHomeSecurityBasicCamera1080p-normal	benign	26707	1265	0.047	0
0	xiaomi-MiLEDCeilingLight-normal	benign	19902	0	0.000	0
0	xiaomi-MiLEDSmartBulb-normal	benign	2612	0	0.000	0
0	xiaomi-MiSmartCompactProjector-normal-2	benign	162895	0	0.000	0
0	xiaomi-MiSmartCompactProjector-normal	benign	25537	0	0.000	0
0	xiaomi-MiSmartSensorSet-normal	benign	42769	0	0.000	0
0	amazon-EchoShow5-normal-2	benign	1367195	0	0.000	0
0	amazon-EchoShow5-normal	benign	1272456	0	0.000	0
0	beagleboard-BeagleBoneAI-normal	benign	367659	0	0.000	0
0	google-ChromecastUltra-normal	benign	131947	282	0.002	0
0	google-Home-normal	benign	82198	0	0.000	0
0	google-HomeMini-normal-2	benign	13760	4071	0.296	1
0	google-HomeMini-normal	benign	83530	2142	0.026	0
0	google-NestHub-normal-2	benign	24902	2104	0.084	0
0	google-NestHub-normal	benign	166627	0	0.000	0
0	google-NestMini-normal-2	benign	22237	6770	0.304	1
0	google-NestMini-normal	benign	150036	1796	0.012	0
0	honeywell-C2WiFiHomeSecurityCamera-normal	benign	47437	0	0.000	0
0	honeywell-W1WaterLeakAndFreezeDetector-normal	benign	196	0	0.000	0
0	leotec-AndroidTvBoxGCX2432-normal-2	benign	200974	0	0.000	0

Table B.3: Classification on benign traffic in VARIoT dataset

C

NURSE article

NURSE: eNd-UseR IoT malware detection tool for Smart homEs

Antoine d'Estalenx and Carlos H. Gañán
Delft University of Technology

ABSTRACT

IoT devices keep entering our homes with the promise of delivering more services and enhancing user experience; however, these new devices also carry along an alarming number of vulnerabilities and security issues. In most cases, the users of these devices are completely unaware of the security risks that connecting these devices entail. Current tools do not provide users with essential security information such as whether a device is infected with malware. Traditional techniques to detect malware infections were not meant to be used by the end-user and current malware removal tools and security software cannot handle the heterogeneity of IoT devices. In this paper, we design, develop and evaluate a tool, called NURSE, to fill this information gap, i.e., enabling end-users to detect IoT-malware infections in their home networks. NURSE follows a modular approach to analyze IoT traffic as captured by means of an ARP spoofing technique which does not require any network modification or specific hardware. Thus, NURSE provides zero-configuration IoT traffic analysis within everybody's reach. After testing NURSE in 83 different IoT network scenarios with a wide variety of IoT device types, results show that NURSE identifies malware-infected IoT devices with high-accuracy (86.7%) using device network behaviour and contacted destinations.

1 INTRODUCTION

With the widespread of appliances that come with network connectivity, more and more devices are being connected to the Internet. This variety of devices led to the emergence of the term Internet of Things (IoT). While this term can be used to refer to all sort of devices, it is commonly used to describe sensors and devices that collect data and send it over the Internet. The IoT is a growing part of the Internet and outgrows non-IoT devices over the Internet. In 2020, the number of connections from IoT devices surpassed the connections from non-IoT devices with about 54% of connected devices being IoT at the end of 2020 [24].

However, not everything about connecting devices to the Internet is beneficial. In fact, the additional communication capabilities create more entry points in the device, enlarging the attack surface and causing more security risks. The lack of security controls in IoT devices could cause sensitive data to be stolen or give the attacker the possibility to remotely control the device [35]. One of the possible misuses of insecure IoT devices is to use them in botnets. Botnets are networks of compromised devices called "bots" to which an attacker can send instructions remotely [22]. These botnets can be used to perform malicious activities such as sending spam, click fraud, hosting of illegal content, or performing Distributed Denial of Service (DDoS) attacks.

Despite the damage that malware-infected IoT devices cause, device owners are not only often unaware of these infections but they may not even know which devices in their home are connected to the Internet [11]. While some have antivirus software to detect malware in their personal computers and laptops, they may not realize that their IoT devices may also be infected. Antivirus software proposes easy to use solutions to detect infections on computers, however there is no such solution for infected IoT devices.

Developing antivirus software for IoT devices is a challenging task [44]. The heterogeneous aspect of the IoT makes it complex to have a software compatible with any device. Also, the limited resources and computer power would set constraints for such a software. However, many researchers have designed solutions to detect malicious traffic with high precision, which could detect the infection looking at the traffic from a device. While these solutions exist, they are not used in homes to help users with their infected devices and users still own devices that are part of botnets unbeknownst to them. IoT users have no available tool to detect the infected devices on their own. In this paper, we study how current malware detection techniques can be applied in IoT scenarios to then create a tool that any home user could use.

Previous studies relied on placing additional hardware to act as firewall in order to block unwanted IoT traffic [8, 12, 23] or modifying the home gateway [25]. All these studies presume that home users have the knowledge and skills to perform complex network-related monitoring tasks; however most Internet users do not have these skills [19]. To cover this gap, we developed NURSE, a tool that enables end-users to detect IoT malware infections without requiring additional hardware nor modifying their network configuration. This tool leverages a rule-based engine to detect anomalies within a user's IoT generated traffic to then signal directly to the user which device is infected and block this traffic.

Our contributions are the following:

- We design and develop NURSE: a tool to detect malware-infected IoT devices within home networks. NURSE does not require neither modifying nor adding new hardware components to the network, becoming the first in-band malware detection system for IoT home networks.
- We identify network-related features that characterize malware-infected IoT devices. These features are integrated within NURSE's detection logic identifying with high accuracy the most prevalent attack vectors used by IoT malware, including horizontal scans, brute-forcing and DoS attacks.
- We evaluate NURSE using a set of 83 different IoT network scenarios including both networks legitimate and malicious traffic. These include a wide variety of IoT device types ranging from air purifiers and weather stations, to alarm systems and android TV boxes. The results show that NURSE is able to detect malware-related attacks with 86.7% accuracy.
- We make NURSE source code available to the community for further extensions and testing (LINK REDACTED). This would allow not only end-user to monitor their network but

Anonymous Submission to CPSIoTSec, Nov 2021, Seoul,South Korea
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

also Internet Service Providers to better assist their infected subscribers with remediation advice.

2 RELATED WORK

Previous research efforts related to IoT malware remediation have mostly focused on addressing two problems: botnet detection and IoT devices identification. In this section, we study botnet detection techniques in order to better understand how botnet detection is performed, and what the detection techniques require to be applicable. Next, we present a thorough set of techniques to identify IoT devices.

2.1 Botnet detection

Botnets are an old security problem and as such the number and diversity of techniques to mitigate this threat is vast. Botnet detection techniques can follow four different approaches [14]: signature-based detection, anomaly-based detection, mining based detection and DNS-based detection.

Signature-based detection: Signature-based techniques are now widespread to detect malicious behavior in a network [9]. They are commonly integrated within Intrusion Detection Systems (IDSs) that monitor traffic to find patterns that correspond to specific activities that reveal that a device is under attack, or that a device is infected by a malware. However, a limitation of signature-based detection is that it can only detect known attacks and cannot identify new threats since their signatures are unknown.

Anomaly-based detection A second approach for botnet detection is anomaly-based detection. This statistical approach tries to identify abnormal events that could be caused by an attack. For example, a device that is part of a botnet could take part in a DoS attack and suddenly send a lot of packets to a single host. Such a behavior could be quite abnormal for the device and an anomaly-based detection method could raise an alert to suggest that the device is acting oddly, suggesting that it may be infected. An extension of this detection method consists in not only the temporal correlation, which helps detecting abnormal behavior at a certain time, but also spatial correlation, to note if many devices perform the same actions at the same time. This approach is presented in BotSniffer [17] where researchers aim at detecting discussions with command and control servers using spatial-temporal correlation. BotSniffer monitors the traffic, mostly keeping track of suspicious activities and use of protocols that are sometimes used to communicate with C&C servers (HTTP and IRC at the time of writing).

Mining-based detection The third approach for botnet detection is based on data-mining to find examples of malicious traffic and use machine learning to identify malicious traffic [6]. Alparlsan et al. in [6] present several examples of data-mining based detection techniques, with techniques analyzing IRC communications to identify suspicious nicknames and servers or flag IRC flows that correspond to command and control communications looking at payloads. However, botnets are evolving and are now using other protocols such as HTTP, and sometimes even custom protocols. So data-mining solutions are now looking at networking flows, and one example of such technique is BClus [15]. This method analyzes the flows, and adopts a behavioral-based detection approach. The method works by analyzing flows, cutting them in time windows to reveal temporally local behavior, and then aggregating them by source IP address to reflect the behavior of each host.

DNS-based detection The final approach for botnet detection consists in inspecting the traffic in a specific protocol to detect malicious behavior [5]. This can be done with IRC or HTTP communications, however, such an inspection could easily be countered by encrypting commands before sending them. One of the most commonly inspected protocols is DNS. This protocol is useful to inspect for two main reasons: first it is not encrypted and second because it is used a lot as it connects the domain names with IP addresses. Inspecting DNS gives the list of domains contacted by the device, and the IP addresses associated with it.

2.2 IoT device detection

2.2.1 Active scanning. A first approach to identify IoT devices in the Internet consists on scanning the whole IPv4 address space, like Shodan [2] does. This scanner allows to scan the internet for device with a specific port or banner and is often used by attackers to find devices that run vulnerable versions of software. The use of scanners to find devices of a certain type is illustrated in [27] where researchers show how Shodan, Masscan and Nmap can be used to find specific IoT models that are vulnerable. Similarly, researchers analyzed the network of CERN, trying to identify IoT devices using web-scraping [3]. They identified all devices and profiled them, first by scanning their open ports, and then by scraping their web interface when they had one. Using this technique, they managed to identify many IoT models and manufacturers. They then performed vulnerability assessment on the identified devices to note than 11% were vulnerable with default credentials, and then another 13% had easy to guess credentials or manufacturer's default credentials.

2.2.2 Passive scanning. A second interesting approach to IoT identification would be to identify IoT devices in traffic captures [20, 28]. Although it is a new research domain, some researchers have developed techniques to identify devices based on traffic captures. Most of these techniques are based on domain names that are contacted by devices since these can be easily obtained from captures, as they are not encrypted in DNS queries. Authors in [33] propose a model to fingerprint IoT devices behind a NAT (Network address translation) and identify them in an accurate and explainable way. Their idea is to profile each device with a list of domains associated with their query frequency. The researchers then compare the fingerprints of the device they want to identify with fingerprints of known devices to identify the unknown device. The method identifies a device based on a previously recorded profile quite accurately. The researchers also note that there is little confusion between IoT devices, except for some IoT models from the same manufacturer, for example Google Home and Google Home mini, which contact the same domains and therefore have identical DNS profiles. Their tool detects most IoT devices, only missing some when they were mixed with other traffic behind the NAT. Finally, the researchers used their tool on an ISP DNS traffic captures and performed identification among the 40 million ISP clients to provide the IoT device distributions. Since researchers cannot own all IoT devices, researchers from Princeton university crowdsourced

IoT identification [20] and proposed *IoT-Inspector*, a tool aimed at collecting device traffic to create a dataset for IoT identification. The tool is meant to be run by volunteering users who own devices on their own computer.

Detecting IoT Devices on the Internet: Hang Guo and John Heidemann [18] attempted to detect IoT traffic to measure the growth of IoT devices. They propose three detection techniques: IP-based, DNS-based and TLS-based. The IP based technique works by listening to DNS traffic of 10 devices they bought to establish profiles of contacted IP addresses. They then set a threshold and assign a label to an unknown device is it share 2/3 of its contacted IP addresses with a known device. This technique works well on their devices, but researchers note that it is not time resilient as IP addresses assigned to domains can change over time. The second technique is DNS-based. It is similar to the previous one, except that working with the domain names instead of IP addresses prevents the changes over time and allows to identify third-party domains and manufacturers domains thanks to domain name and WHOIS information. The techniques show great accuracy over their devices, and seem to be more time resilient since it detected devices in several years old traffic captures. Finally, the TLS based detection method attempts to identify IoT devices that provide an HTTPS interface (such as IP cameras). Researchers analyze the TLS certificates of the web page and search for keywords that could identify the manufacturer or device type. These three techniques show accurate detection in the experiments carried by the researchers; however, they are limited to detecting the device that were in the training set.

Detecting IoT Devices at an ISP level: The previous study mostly performed detection on traffic captures at local scales, for example on university campuses. Authors in [36] studied the detection of IoT devices at a higher level: moving the detection at the ISP (Internet Service provider) or IXP (Internet exchange point) level. The detection technique relies on detecting IoT-specific infrastructures. The researchers first identify domains contacted by devices by monitoring DNS traffic and classify them into IoT specific domains and generic domains. They then obtain the IP addresses associated with IoT-specific domains using DNSDB [37], and filter out the ones that are shared between multiple services to obtain the dedicated infrastructure. Finally, the endpoints (IP address, port) are associated to the device which creates a profile for each device type. This method achieves great detection performances when evaluated, however, researchers acknowledge that it is limited as it cannot identify devices which were not in the training set, and does not work well for devices that have limited network traffic.

3 NURSE: A TOOL FOR END-USERS TO DETECT IOT MALWARE INFECTIONS

Current state-of-the-art malware detection techniques were not devised to be used by end-users but instead by network operators. With the increase of Internet-connected devices at home, end-users have inadvertently become operators of a network of heterogeneous devices with different network interfaces and communication protocols. Besides the fact that the great majority of these users do not have the technical skills to manage the security of their networks, they also lack tools to monitor these networks. NURSE is a tool tailored specifically to tackle this issue by alerting IoT users when their devices have been victims of IoT malware.

To reach this goal, NURSE is designed following a modular approach. Our tool is composed of multiple modules:

- *ARP spoofer*: the process that performs the ARP spoofing. By performing a double ARP spoofing, impersonating both the router and a device in the network, NURSE can receive all the packets from the device to the router and from the router to the device;
- *IoT packet filter*: filters packets belonging to the IoT devices to be monitored;
- *Flow extraction and protocol parsers*: Extracts the flows and parses the packet content;
- *Traffic monitoring*: monitors the information from the packet data, updates a database regularly;
- *Traffic Analyzer*: Analyzes the database per time window to then score each domain name as well as IP address;
- *Alert generation*: raises alerts taking into account the user's configuration, traffic history and output of the traffic analyzer;
- *GUI*: shows the alerts to the user via a local web server.

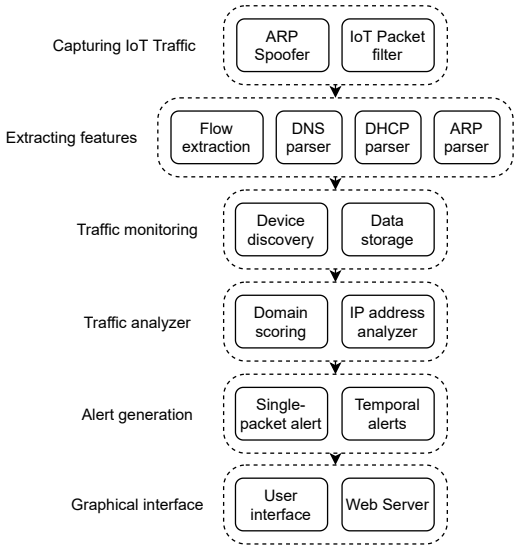


Figure 1: NURSE modular architecture

The tool architecture is shown in Figure 1, with boxes showing the main modules and their purposes, and arrows showing how the data (packets, alerts...) flows between the modules. The tool uses a multi-threaded approach, as most of these modules have to run simultaneously.

3.1 Capturing IoT traffic

The traffic from the IoT devices is intercepted using ARP spoofing [43]. The ARP spoofer runs an infinite loop in which it sends ARP spoofing packets periodically to ensure that the ARP spoofing is maintained. Running it as a parallel thread also allows us to send a `QUIT` signal from the Host state thread to the ARP spoofing thread

when the program is closed. When this signal is received, the ARP spoofer thread undoes the ARP spoofing by sending ARP packets with the actual IP and MAC addresses to all the devices it spoofed. This ensures that the device which were spoofed can find the actual gateway immediately. Otherwise, our tool may cause a network disruption until the spoofed devices update their ARP tables and find the real MAC address of the gateway.

The packet capture is performed using `Scapy` [10], which allows our tool to capture packets at the interface level. Therefore, our sniffer captures all of these packets. Then, we filter out packets captured at the interface level are relevant for our traffic analysis.

First, we filter the packets based on the Ethernet layer, and more precisely based on the source MAC address. This is to prevent the host from capturing the packets it is sending. The packets sent by the host can be: (i) ARP packets sent by the ARP spoofing thread; (ii) packets that are forwarded from the victim to the gateway to keep the traffic flow uninterrupted despite the man-in-the-middle position; or (iii) actual traffic from the host, for example if the user uses another application while our tool is running.

The first type of packets are generated by our tool and are therefore not relevant to analyze. Similarly, the forwarded packets are redundant because they are sent once we have intercepted the original packet. Therefore, scanning the forwarded packets would cause every packet to be counted twice.

Capturing the third category of packets from our host could prove useful to analyze the traffic of our own host. This could allow our tool to perform botnet detection on the host it is running by analyzing its outgoing traffic. However, we did not design our tool with this use case in mind, and the main use case is to monitor other devices than the running host. There exist more specific software dedicated to malware detection on the machine such as antiviruses, that also exploit the fact that they are running on the analyzed machine and can for example by analyzing files looking for malware signatures.

Therefore, the captured packets are from and to the devices that are specified by the user. We drop all other packets, to prevent interference from the running host.

3.2 Extracting features

Once the packets have been filtered to keep only the ones that are sent or destined to one of the pre-selected IoT device, the packets are parsed in the packet parser module. The packets are then parsed depending on their content and the layers identified by `Scapy`. First, for each TCP or UDP packet, we extract its *flow* to group packets per flow in our database. A flow consists of (i) a source IP address, (ii) a destination IP address, (iii) a source port, (iv) a destination port; and (v) a transport protocol: UDP or TCP. With this flow-based approach, we can group all packets that go from a port to another. We then keep the timestamp, flags, byte size of the payload and direction (inbound or outbound) for each packet. These features are the ones that are sent to the traffic monitor module to be added to the flow database. ARP spoofing does not allow the spoofer to decrypt traffic, which is why we do not collect more general features on the payload, since it can be encrypted.

DNS parsing. For DNS packets, we parse the DNS payload as well as the UDP header, which allows us both to analyze the flow and the exchanged bytes, and the content of DNS communications. The detection of DNS contents in the packets relies on the flow ports since DNS is recommended to be used with UDP port 53 in RFC1035 [31]. We also parse the DNS responses to extract the answer records. We especially parse the `A` and `AAAA` records to obtain the IP addresses associated with the queried domain.

ARP parsing. As ARP packets are not encrypted, we can also parse their content. We parse these packets to keep track of the MAC and IP addresses of the devices in the network. These associations are stored in an ARP table mapping IP addresses to MAC addresses. This table is stored by and is accessed by our threads when they need to obtain MAC addresses of devices. For example, the ARP spoofer is written in such a way that it queries for the MAC address of the targeted devices if it is unknown. However, if a packet that announces the MAC address of a device is intercepted (for example because it was broadcasted to all devices), this association is registered so that the ARP spoofer does not need to query the MAC address again.

DHCP. DHCP helps with IP addresses management and for keeping track of which devices are in the local network. DHCP packet parsing is used to grab the names that are advertised by devices in the local network. This is useful to present some readable names to the user when he looks at the list of detected devices.

In our packet parser, we parse DHCP packets to obtain advertised host names. We then build a database associating the device names to MAC addresses. Combining this with the ARP table allows us to convert an IP address to the advertised host name, which will be useful when presenting data to the user or when listing devices for the selection of monitored devices.

3.3 Traffic monitoring

The traffic monitoring module is in charge of two main tasks: (i) keeping a database with all the with all the features extracted from parsing DNS, ARP and DHCP packets; and (ii) discovering new IoT devices. This database can be seen as a passive DNS database: a database that keep tracks of IP associated with domains, the device names and their corresponding MAC addresses. This database is built live as the tool runs based on the DNS packets intercepted. This allows us identifying which domain are contacted by the device in order to keep track of the contacted domain, and to map the flows IP addresses to domains. This database also records the timestamps of the DNS queries per domain and per device, to keep track of the queries each time they happen instead of merging them per queried domain.

Besides maintaining the database, the traffic monitoring module leverages ARP packets to discover new IoT devices. However, using only ARP can be limited to detect new hosts because hosts usually only advertise their IP address via ARP as they join the network. Therefore, unless the gratuitous ARP packet sent upon joining the network is captured, the only way to discover new hosts via ARP is to perform a live scan of the network. Therefore, the monitoring module also detects when a packet reveals the IP of an unknown device. Of course, we cannot intercept traffic of an unknown device since the ARP spoofer only targets devices that have been selected by the user from the list of detected devices. However, we can detect

new devices when a targeted devices communicate with them, and therefore specify their IP as a destination. Upon detection, NURSE sends ARP packets to check if the IP address exists, obtains the MAC address and the device name of this newly detected device and adds these to the databases, so that the device is now detected and available to monitor for the user. When parsing capture files, this part is disabled, but the newly detected device is automatically added to the list of monitored devices so that the tool parses packets from and to this host.

3.4 Traffic analyzer and Alert system

This module leverages the features stored in the database to flag IoT infections based on the packets related to IoT traffic. We consider two types of alerts: alerts that flag suspicious single packets and alerts that flag events over a time window.

3.4.1 Single packet alerts. These are usually not attacks in themselves, but suspicious packets that could reveal that an attack is performed. In particular, alerts are raised under three conditions: (i) Spoofed source IP address, (ii) contacting an IP address without prior DNS query (Hard coded IP); and (iii) contacting a blocked IP address.

Spoofed source IP addresses: NURSE flags packets in which the source IP address field has been filled with an incorrect address. When a device sends a packet from a home network, the expected source address is either the local IP address in the network, or the public internet address if the packet is sent over the internet. If the address in the source IP field is none of these two, the packet is flagged.

IP spoofing is not an attack in itself, and can even be used in legitimate situations, for example to simulate packets coming from multiple hosts with all sent from a single host. Attackers may also use it to bypass IP filtering in a network, however, its main malicious application is DDoS attacks. In DDoS attacks, the attacking device sends many packets but does not intend to receive and deal with the replies. Therefore, DDoS attackers spoof the source IP field so that the victim sends replies to random internet endpoints where these will be discarded, instead of flooding the attacking device with replies.

A spoofed IP alert may not be a proof that a device is infected in itself, but multiple alerts of this type, combined with alerts that flag DoS activity could definitely reveal that the device is taking part in DoS attacks, and could therefore be infected with malware.

Hardcoded IP addresses: Previous research has shown that IoT malware includes hardcoded IP addresses to contact the C&C and to spread the infection [41]. When an IoT botnet attempts to find other vulnerable hosts, for example via port scanning, it usually contacts random endpoints or a list received from its C&C server. In this case, the device contacts a lot of devices directly via their IP address with no previous DNS query, which could be flagged by this alert. On the contrary, it is now quite rare that a user would contact an IP address directly without first querying a DNS resolver. Thus, NURSE triggers an alert if this network behavior is observed.

We however note that contacting an IP with no previous DNS query is not a malicious behavior. There could be plenty of usages where a device contacts an endpoint directly via its IP address, for example if no domain was registered for this IP address. Another limitation of this alert is DNS caching. To avoid querying DNS each time a domain is contacted, hosts usually save the DNS responses for a small time when the response is first received. Then, when a domain is queried for the second time, if the domain is already in the DNS cache, the device contacts the saved IP addresses directly. Therefore, if our tool is launched on a device that has cached some DNS entries, requests to the cached domains will appear to be sent to hard coded IP addresses until the cached DNS responses expire, which would raise some alerts on benign traffic.

Low reputed IP addresses: The final single packet event that NURSE flags is contacting an IP address that is included in any of the preconfigured reputation blocklists (RBL). There exist multiple IP-based RBLs that contain IP addresses which have been flagged for being associated with malicious activities.

In its initial release, NURSE uses the RBLs provided by Spamhaus [39]. This project provides lists of domains and IP addresses that have been detected as senders of bulk email, or involved in other malicious activities. However, they also provide the *Spamhaus Exploits Block List (XBL)* which contains IP addresses of hosts that have been infected with malicious third-party exploits. We therefore check if the devices we monitor communicate with IP addresses of the XBL database, which could reveal that they are communicating with their C&C server or with other devices in the botnet that have been previously detected as sending bulk email or performing malicious activities.

Spamhaus allows the IP addresses to be checked via a public DNS server. For example, the IP address 1.2.3.4 can be checked by sending a DNS query for the domain *4.3.2.1.zen.spamhaus.org*. The returned IP address indicates whether the queried IP address is in Spamhaus blocklists. Therefore, whenever our tool intercepts traffic going towards a new IP address, it checks if this IP address is in Spamhaus blocklists.

Domain name scoring: NURSE leverages a random forest classifier to score each domain name as seen in the captured IoT traffic. The classifier leverages the following thirteen features: (1) top-level domain, (2) total domain length, (3) number of subdomains, (4) mean of subdomains length, (5) consonant ratio, (6) consecutive consonant ratio, (7) digit ratio, (8) repeated characters ratio, (9) Shannon's entropy, (10) mean of trigram frequencies, (11) standard deviation of trigram frequencies, (12) ngrams matching with English, and (13) word count.

The classifier is trained with malicious domain dataset from 360 DGA list from Netlab [29]. The dataset of malicious domains contains about 1,500,000 domains from 50 DGA families. As benign dataset, we used two domain lists: the top 1M domains from majestic million [26], and lists of random domains that were uploaded by OpenDNS on GitHub in 2014 [30]. As most of the top domain names are in English, we also get the top 50 domains from each country which are displayed on Alexa [4], to add some linguistic diversity in the training set. While OpenDNS specifies on their GitHub repository that they removed the domains they detected as suspicious from their domain datasets, we cannot be sure that all the top domains are actually benign domains. Therefore, we use the `pysafebrowsing` [42] package to analyze the domain list with

Google Safe Browsing API [16], which allows us to check whether a website can be trusted or not according to Google. Doing so, we find about 1,250 malicious domains, most of them coming from the majestic million list and being flagged as SOCIAL_ENGINEERING by Google. We therefore remove these from the benign set, but do not add them to the malicious dataset since they are not linked to malware activities, which we try to flag here.

To select the features we want to use, we implemented some features from previous work and added some of ours, then we selected the ones we wanted to keep based on how well they separate benign from malicious domains and how practical they were for our model. One example of an impractical feature was the domain registration. Checking whether the domain is actually registered is a good indicator of whether the domain is benign or is generated by a DGA, since DGA generate a lot of domains of which only one is registered for a short time, while most benign domains are registered to support applications or websites and therefore are meant to stay available for a long time. However, checking whether a domain is registered or not requires querying a DNS server which is too slow and impractical for our tool: this would take up quite some bandwidth from the user of our tool. Also, such a feature would be redundant since our tool intercepts the traffic of the IoT device and therefore can intercept a DNS response and check whether the queried domain is registered or not.

In Figure 2, we present the distribution of domains for features 2 to 13 for benign and malicious domains. This shows that even if the distributions overlap, which shows that some malicious domains may be confused with benign domain, the features we selected split the classes. For example, the word count reveals that benign domains mostly contain 1,2 or 3 words, while most malicious domains have more than 5. While some features show low correlation with each other (such as 2 and 4 since domains rarely have a lot of very short subdomains), the majority of the features are independent from each other. We simply note that the length and subdomain length is correlated with many lexical and statistical features, which is normal given that the longer the domain, the more complex it is likely to be, which therefore increases the features linked to character distribution and repetition such as entropy, 3 grams distribution or repetition ratio.

The confusion matrix of our classifier is shown in Figure 3. The accuracy of the classifier is about 96.9%. This is a satisfying result considering that we had to limit the features to only lexical features. We however note that the false positive ratio is 3.04%, which could cause some benign domains to be classified as malicious when the tool is used by users. We limit the impact of false positives when designing the rules for DGA alerts.

The use of DGA can be flagged in two ways: (i) by noticing that the devices contacts consecutively multiple non-existent domains; and, (ii) by analyzing the domains that are queried. As the domains are generated from an algorithm, they can follow a pattern which could be identified.

The classifier is integrated to the tool as a domain scorer. Given a domain name intercepted in the traffic (in a DNS query), we compute its features and use our random forest classifier to compute the probability of being malicious. This probability is multiplied by 10 to show a score between 0 and 10 to the user and the domain score is logged. The domain list with their scores is displayed to

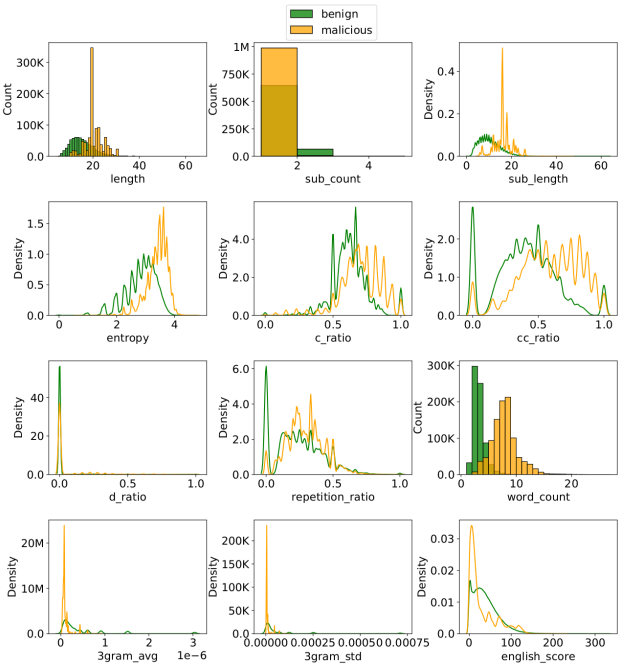


Figure 2: Density distribution of training domains for selected features

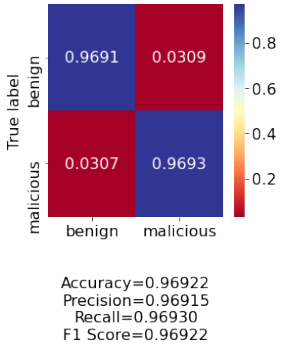


Figure 3: Confusion matrix of NURSE's domain classifier

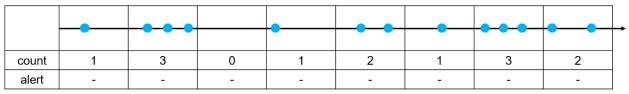
the user which allows him to see which domains are contacted by the device, and how suspicious they look. However, since there may be some false positives, the traffic analyzer analyzes the list of queried domains in time windows and checks if the number of domains with a bad score exceeds a threshold. In that case, an alert for suspicious domains is raised.

3.4.2 Temporal alerts. These alerts are events that are detected over a time window. The traffic analyzer runs through the packets of a time window and detects whether these packets match detection rules of malicious events. The alerts include the following: (i) Denial of Service; (ii) vertical port scanning; (iii) horizontal port scanning; (iv) bruteforce attempt; (v) NXDOMAIN rate; and (vi) DGA domains.

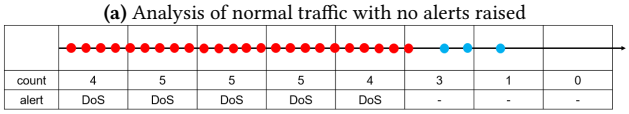
We pick a time window of 1 minute. This time window is to ensure that short attacks such as small port scan or DGA running with small algorithms can be detected, but is also not too short to avoid generating too many alerts that correspond to the same event. Taking one-minute-long time window also allows to smooth

out the traffic spikes, which prevents from flagging any spike as a DoS attempt. Longer time windows could be chosen, but could be exploited by botnets to avoid detection, for example by running an attack for only half the window, which could still disturb the targeted service. The time window duration and thresholds are initially set in the configuration file, they can be modified by the user in the settings panel of the application. The thresholds presented for each alert were set based on observation of benign and malicious traffic. For this task, we used captures from malware captured in a honeypot and benign traffic from IoT devices [7, 34, 38].

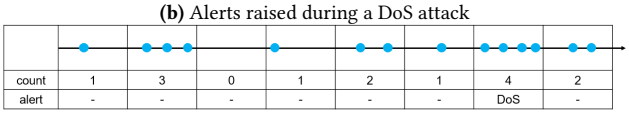
Figure 4 presents how the analyzer splits into time windows and counts the packets or events within each window. It then compares the count of events within each window with the threshold (set to 4 in this example for simplicity). The figure displays 3 examples. In Figure 4a, the traffic is normal and the threshold is never exceeded, leading to no alerts being raised. In Figure 4b, a denial of service attack is happening (labeled with the red dots) and alerts are raised for each time window in which the threshold is exceeded. Finally, the Figure 4c shows how normal traffic could exceed the threshold in a time window and therefore raise an alert that is a false positive.



(a) Analysis of normal traffic with no alerts raised



(b) Alerts raised during a DoS attack



(c) Analysis of normal traffic with a false positive

Figure 4: Example of traffic analysis (with threshold set to 4)

Denial of Service: To detect denial of service attacks, NURSE set a threshold of maximum connections towards a single port of an IP address within a time window. The analyzer gathers all opened connections per flow key. It then computes the number of connections opened in each time window. For TCP packets, the connection count is obtained by counting the number of SYN packets sent to the port. For UDP packets, the number of connections opened is simply the number of packets sent since UDP is a connectionless protocol.

The opened connections are then gathered per key depending on the event we target. Since a denial of service corresponds to a single host opening a lot of connections with a targeted port on a single host, the counts are gathered with keys: (source IP, destination IP, destination port).

Then the counts per time window are compared with a threshold that corresponds to a suspiciously high number of connections for a time window. This threshold has a default value of 120 packets per minute which was set based on observations from IoT malware captures.

Vertical and Horizontal Port scanning: The detection of port scanning follows the same implementation as the denial of service, except that it gathers connection with different keys. For horizontal port scanning, flows are gathered with keys (source IP, destination port), for vertical port scanning, the keys are (source IP, destination IP). If the count per key exceeds a pre-configured limit per minute, an alert for port scanning is raised.

An important point is that horizontal port scanning has to whitelist some services. Otherwise, a user connecting to multiple websites would be flagged as connecting to too many IP addresses on port 80 (HTTP) or 443 (HTTPS). We therefore add a whitelist of ports to avoid raising alerts on these ports. No other ports are added to this whitelist as based on the captures we observed, IoT devices often communicate via HTTP. Some devices may use more specific protocols and other ports, but these are not likely to be used by many different endpoints, which should not raise horizontal port scans alerts. For users who note that their device uses a specific protocol a lot and have port scans alerts raised on normal traffic, we offer the possibility to add custom ports to the ports whitelist.

Bruteforce attempts:

To detect bruteforcing, we use the same approach as in denial of service detection, except that the destination ports are merged if they correspond to SSH or telnet. The goal is to detect multiple connections that would correspond to multiple passwords guesses. The threshold is set so that a device that opens 5 connections in a minute is flagged as bruteforcing the service. Even considering that SSH allows 6 authentication failures before disconnecting [1], bruteforcing while evading the connection would therefore only reach 30 guesses per minute, which is slow for bruteforcing.

This technique only relies on the number of connections. More precise approaches can be taken, especially for Telnet which is not encrypted, but would require to analyze the packets content which could be invasive and would only be possible for non-encrypted protocols. It can however be noted that some advanced network monitoring tools attempt to detect SSH bruteforcing in encrypted traffic [40]. We however did not implement this technique as only works in specific cases and requires monitoring the connection status precisely.

DGA domains: We already detailed how DGA domains are detected thanks to a classifier in the previous section. We, however, do not raise an alert immediately when a domain is flagged by the classifier, instead DGA domains are set up as a temporal alert to prevent false positives.

As the classifier is not perfect and may classify benign domains as false positives, we do not raise an alert for each domain classified as DGA. A temporal approach also suits DGA detection because DGA are designed to contact many domains in a short amount of time to find the rendezvous point. That is why, a machine running a domain generation algorithm would be flagged even with a temporal alert. To prevent the DGA from being successful or evading detection, we set the threshold at a low number of domains per time window. The default value is 5 domains flagged as malicious per minute. This prevents false DGA alerts as it would require that the classifier

raises 5 false positives in less than a minute. However, it also makes it harder to evade detection, as 5 domains per minute means that the DGA should take at least 12 seconds per domain, which is a slow rate and could cause the DGA to find the selected domain in a long time, which partially defeats the purpose of DGA.

NXDOMAIN rate: This alert is simply raised when one hosts sent a number of DNS queries that returned a `NXDOMAIN` (non-existent domain) code that exceeds a certain threshold. This alert complements the alert for malicious domains and allows to flag DGA even if the classifier misses the domains as malicious, since the host that runs the DGA will certainly contact many non-existing domains before finding the actual meeting point.

However, generating `NXDOMAIN` is not always due to malware, they could be generated by a user making a typo in a domain, or to check that the domain name server is working correctly, as does the Chrome browser on startup to detect DNS hijacking [13]. The chosen threshold is 5 `NXDOMAIN` per minute, which should prevent some of these false positives since Chrome only queries 3 non-existent domains on startup.

4 EVALUATION

4.1 Testbed

We test NURSE leveraging IoT network captures including both benign as well as malicious traffic. We used two datasets: (i) a labeled dataset with malicious and benign IoT network traffic from CTU IoT-23 [32]; and (ii) captures of IoT traffic from the VARIOT projects [35]. In total, we used 83 scenarios, 23 from the IoT-23 dataset, and 60 captures of normal traffic from the VARIOT dataset.

The CTU IoT-23 dataset contains captures of botnet activities running on IoT devices. The dataset includes 3 devices: an Amazon Echo, a Phillips Hue and a Somfy door lock device. The devices were infected with IoT malware such as Mirai or Okiru, and traffic was captured. The researchers then labeled the flows manually indicating whether these were parts of horizontal scans, attacks, C&C communications, DDoS file downloads, benign traffic, etc. We used these flows classification to label scenarios with the major type of malicious traffic that can be observed in the capture. These labels are presented in the column "Type" in Table 1.

The second dataset contains captures of compromised devices and normal traffic. It is important to note that even if some captures correspond to DoS attacks, and firmware changes which could be relevant for NURSE, most captures correspond to an attacker controlling the device remotely. In these captures, an attacker controls a device via an exploit, and performs basic actions. For example, when attacking a smart bulb, the attacker can turn the device off, adjust the brightness, etc. We note that while these attacks are not benign, they are not botnet attacks and are mostly involve normal actions, except that these are triggered remotely by the attacker. For this reason, we do not evaluate our tool on the captures of compromised devices, but we use the captures of benign traffic to check if alerts are raised on non-infected devices.

4.2 Results

The summary of the detection results are shown in Figure 5. NURSE achieved an accuracy of 86.5% when evaluated over the 83 scenarios, i.e., it successfully detected all the attacks related to IoT malware. Only 5 scenarios that only contained C&C communication created false negatives. Similarly, the rate of false positives is also quite low (7.23%) and could be decrease to zero if NURSE would activate the option to whitelist `QUIC`-related packets.

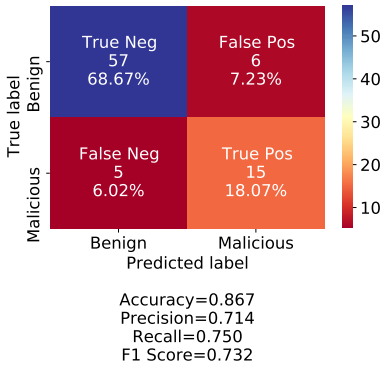


Figure 5: Confusion matrix over the evaluation data

Table 1 presents the alerts raised on the 23 scenarios of the IoT-23 dataset [15]. NURSE detected and raised alerts for all the scenarios were malicious traffic was present. The alerts that are not present in Table 1 were not raised during the analysis. We first note that all horizontal scans and DDoS events have caused the corresponding alerts to be raised which shows that detection for these events work on real attacks. Therefore, NURSE detects real IoT botnet attacks with 100% accuracy.

However, we note that some scenarios such as *CTU-Malware-20* and *CTU-Malware-21* with malicious traffic did not raise any alerts. These scenarios only contain communications with a C&C server before the attack takes place. Looking in detail at the results, we note that the malicious domain that was contacted *top.haletteompson.com* was not flagged as DGA domain and that the corresponding IP was not blocklisted. The communications then happened over port 443 and were encrypted using SSL. As we cannot decrypt encrypted communications in ARP spoofing to perform deep packet inspection, NURSE no possibility to detect that this communication was malicious if the domain and associated IP were not classified as malicious.

We finally note that no alerts were raised on the benign traffic captures which shows that there were no false positives in this dataset. However, the evaluation on some benign captures of the VARIOT dataset reveals that some devices raised alerts in benign traffic (see Table 2). While most of the devices have almost no alerts raised in benign traffic, we note that some devices from Netatmo, Google and Xiaomi raised DoS alerts on benign traffic. Those alerts were mostly triggered by exchanges over UDP where a lot of small packets were sent in a short time window. Such a behavior for example, is quite common in Google products as Google developed QUIC protocol over UDP to send data [21]. Currently, as IoT malware is not leveraging QUIC, NURSE allows to whitelist QUIC-related flows to eliminate these false positives.

4.2.1 Traffic speed impact. We tested the impact of running NURSE on the network throughput in a controlled environment. The test-bed consisted of an Android device running `iperf` to

Table 1: Alerts raised on IoT-23 dataset

Scenario	Type	# Packets	Alert type		
			H. port scan	DoS packets	Hardcoded IP
CTU-Malware 7	C&C, H scan	≈2,500,000	2,505,201	0	2,506,411
CTU-Malware 1	C&C, H scan	1,686,291	377	1,700	346,590
CTU-Malware 17	C&C, DDoS, H scan	≈2,500,000	982,507	0	1,267,942
CTU-Malware 20	C&C	50,156	0	0	0
CTU-Malware 21	C&C	50,277	0	0	0
CTU-Malware 3	C&C, H scan	496,959	72,362	0	74,915
CTU-Malware 33	C&C, H scan	≈2,500,000	1,963,656	0	2,533,130
CTU-Malware 34	C&C, DDoS	233,865	0	120,546	778
CTU-Malware 35	C&C, H scan	≈2,500,000	708,317	0	708,342
CTU-Malware 36	H scan	≈2,500,000	2,499,346	0	2,499,609
CTU-Malware 39	C&C, H scan	≈2,500,000	2,518,641	0	2,518,641
CTU-Malware 42	C&C	24,485	0	0	0
CTU-Malware 43	C&C, H scan	≈2,500,000	2,537,578	0	2,537,512
CTU-Malware 44	C&C	1,309,350	0	1,325,498	4
CTU-Malware 48	C&C, H scan	≈2,500,000	353,207	0	353,207
CTU-Malware 49	C&C, H scan	≈2,500,000	842,193	0	842,261
CTU-Malware 52	C&C, H scan	≈2,500,000	1,267,307	0	1,267,306
CTU-Malware 60	C&C, DDoS	≈2,500,000	1,267,307	2,541,029	21
CTU-Malware 8	C&C	23,623	0	0	2,743
CTU-Malware 9	H scan	6,437,837	6,411,363	146	7,523,827
CTU-Honeypot 4	benign	8,077	0	0	1
CTU-Honeypot 4-1	benign	21,664	0	0	0
CTU-Honeypot 5	benign	397,245	0	0	0
CTU-Honeypot 7-1	benign	6,802	0	0	0
CTU-Honeypot 7-2	benign	119,638	0	0	0
CTU-Honeypot 7-3	benign	62,851	0	0	0
CTU-Honeypot 7-4	benign	121,554	0	0	0
CTU-Honeypot 7-5	benign	121,566	0	0	0
CTU-Honeypot 7-6	benign	119,158	0	0	0

measure available bandwidth. We first performed a test without the NURSE tool running, then started the tool, monitored the android device and performed a speed test again to measure the difference. We repeated this experiment 20 times.

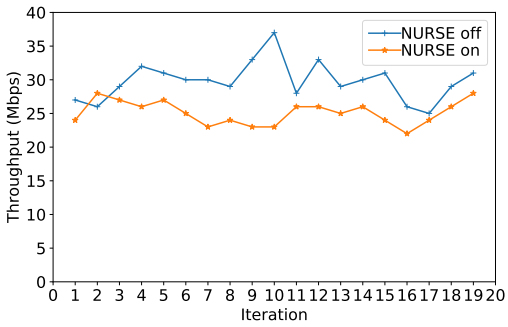


Figure 6: Network speed tests with and without interception

During these experiments, the average speed was 30.46Mbps without the tool running, the graph of the test is shown on Figure 6. When the tool ran, we observed that it started with a very slow connection of less than 1Mbps, but after a few seconds, the connections went back up to normal speeds. This can be observed in Figure 6 where the first dot is at about 20Mpbs, before joining back to normal values.

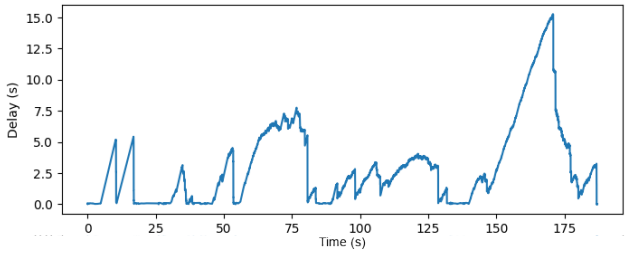
We investigated this issue and discovered that it is caused by the buffering of our packet processing. As we capture packets, the processing of the packet data may take longer than the time before the next packet. Therefore, packets are buffered and we can notice a delay between the time the packet is sent and the time it has been processed. The delay is mostly caused by analysis on new domains or IP addresses which require additional checks (scoring with domain classifier, blocklist check...). We notice that, in periods of high activity, the delay may accumulate and exceed 10 seconds. Although this does not considerably affect the botnet detection (since packets are still processed, only with a delay), it can cause some requests to time out, and cause the device to believe that it is offline. This packet delay however can reduce itself after a while as shown in Figure 7. We note that the delays peaks can reach higher values in Windows, possibly due to the packet capture method on Windows, which relies on an additional dependency.

Since the delay is mostly caused by processing the new domains and new IP addresses, it shrinks to normal values once the checks on new endpoints are performed. That is why, when performing network speed tests, our software always starts with really slow speeds, and reaches higher speeds after a while. Figure 6 shows this behavior, with the first points being slower than the rest. We also noted that some speed tests noted very high latency values that were due to this delay.

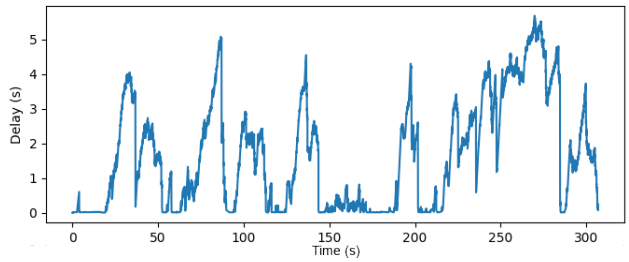
With real web-browsing under monitoring, we observe that websites take longer to load especially for images when these are

Table 2: Alerts raised on VARIoT dataset

Label	Scenario	Attack Type	# Packets	Alert type			
				DGA	Spoofed IP	Hardcoded IP	DoS packets
malicious	beagleboard-BeagleBoneAI-compromised	Firmware change	542,086	0	2	0	0
malicious	google-ChromecastUltra-compromised	Remote Control	27,641	0	0	2	2,129
malicious	google-Home-compromised	Remote Control	8,518	0	0	2	1,423
malicious	google-HomeMini-compromised	Remote Control	8,196	0	0	2	1,687
malicious	google-NestHub-compromised	Remote Control	13,081	0	2	2	2,648
malicious	google-NestMini-compromised	Remote Control	48,642	0	0	2	1,344
malicious	leotec-AndroidTvBoxGCX2432-compromised	Install Trojan	36,784	0	2	1	5,537
malicious	mikrotik-RB951Ui2HnD-compromised-2	Remote control	13,676	0	8	2	0
malicious	mikrotik-RB951Ui2HnD-compromised-3	DoS	31,402	0	0	0	0
malicious	mikrotik-RB951Ui2HnD-compromised-4	DoS	363,433	0	363,122	1	0
malicious	mikrotik-RB951Ui2HnD-compromised	Remote control	19,782	0	0	1	0
malicious	raspberry-pi4modelb-compromised-2	Remote control	81,123	0	2	0	0
malicious	raspberry-pi4modelb-compromised	Remote control	62	0	3	0	0
malicious	tplink-hs110-compromised	Remote control	1,966	0	0	1	0
malicious	tplink-tapoC200-compromised	Remote control	1,468	0	97	1	0
malicious	tplink-tapoL510E-compromised	Remote control	422	0	0	1	0
malicious	xiaomi-MiAirPurifier3H-compromised	Remote control	334	0	0	0	0
malicious	xiaomi-MiHomeSecurityBasicCamera1080p-compromised	Remote control	1,015	0	0	0	0
malicious	xiaomi-MiLEDCeilingLight-compromised	Remote control	255	0	2	0	0
malicious	xiaomi-MiLEDSmartBulb-compromised	Remote control	307	0	0	0	0
malicious	xiaomi-MiSmartSensorSet-compromised	Remote control	1,529	0	4	1	0
normal	amazon-EchoDot3rdGen-normal-2	-	9,848	0	0	0	0
normal	mikrotik-RB951Ui2HnD-normal-2	-	104	0	10	1	0
normal	mikrotik-RB951Ui2HnD-normal	-	6,004	0	2	1	0
normal	mobvoi-TicWatchE2-normal-2	-	16,122	0	3	2	2,462
normal	mobvoi-TicWatchE2-normal	-	89,622	0	11	14	341
normal	netatmo-HealthyHomeCoach-normal	-	2,118	0	0	2	0
normal	netatmo-SmartAlarmSystemWithCamera-normal-2	-	10,470	11	4	1	1,730
normal	netatmo-SmartAlarmSystemWithCamera-normal-3	-	15,446	11	0	1	2,214
normal	netatmo-SmartAlarmSystemWithCamera-normal	-	5,345	20	4	1	639
normal	netatmo-SmartHomeWeatherStation-normal-2	-	1,149	0	4	1	0
normal	netatmo-SmartHomeWeatherStation-normal-3	-	2,987	0	0	0	0
normal	netatmo-SmartHomeWeatherStation-normal	-	3,193	0	3	0	0
normal	netatmo-SmartSmokeAlarm-normal	-	24,121	0	3	1	234
normal	raspberry-pi4modelb-normal-2	-	240,745	0	1	0	0
normal	raspberry-pi4modelb-normal-3	-	74	0	3	0	0
normal	raspberry-pi4modelb-normal-4	-	194,930	0	1	0	0
normal	raspberry-pi4modelb-normal	-	112	0	6	0	0
normal	smarter-Coffee2ndGeneration-normal-2	-	866	0	0	0	0
normal	smarter-Coffee2ndGeneration-normal	-	719	0	3	0	0
normal	smarter-FridgeCam-normal	-	40,691	0	12	4	0
normal	smarter-iKettle-normal-2	-	308	0	2	0	0
normal	smarter-iKettle-normal	-	3,598	0	6	0	0
normal	tplink-hs110-normal-2	-	1,769	0	6	0	0
normal	tplink-hs110-normal	-	1,997	0	5	0	0
normal	amazon-EchoDot3rdGen-normal	-	18,861	0	6	1	0
normal	leotec-AndroidTvBoxGCX2432-normal	-	92,147	0	2	1	0
normal	netatmo-SmartRainGauge-normal	-	1,446	0	7	0	0
normal	tplink-tapoC200-normal-2	-	3,818	0	203	1	0
normal	tplink-tapoC200-normal	-	2,663	0	1	1	0
normal	tplink-tapoL510E-normal-2	-	838	0	0	1	0
normal	tplink-tapoL510E-normal	-	2,670	0	0	1	0
normal	xiaomi-MiAirPurifier3H-normal	-	10,049	0	1	1	0
normal	xiaomi-MiBoxSGlobalVersion-normal-2	-	171,826	0	2	1	4,361
normal	xiaomi-MiBoxSGlobalVersion-normal	-	720,829	0	3	1	146
normal	xiaomi-MiHomeSecurityBasicCamera1080p-normal-2	-	3,267	0	4	1	0
normal	xiaomi-MiHomeSecurityBasicCamera1080p-normal	-	26,707	0	6	5	1,265
normal	xiaomi-MiLEDCeilingLight-normal	-	19,902	0	10	3	0
normal	xiaomi-MiLEDSmartBulb-normal	-	2,612	0	0	1	0
normal	xiaomi-MiSmartCompactProjector-normal-2	-	162,895	0	2	1	0
normal	xiaomi-MiSmartCompactProjector-normal	-	25,537	0	3	1	0
normal	xiaomi-MiSmartSensorSet-normal	-	42,769	0	3	0	0
normal	amazon-EchoShow5-normal-2	-	1,367,195	0	2	0	0
normal	amazon-EchoShow5-normal	-	1,272,456	0	0	0	0
normal	beagleboard-BeagleBoneAI-normal	-	367,659	0	3	0	0
normal	google-ChromecastUltra-normal	-	131,947	0	1	2	282
normal	google-Home-normal	-	82,198	0	1	1	0
normal	google-HomeMini-normal-2	-	13,760	0	12	0	4,071
normal	google-HomeMini-normal	-	83,530	0	1	1	2,142
normal	google-NestHub-normal-2	-	24,902	0	2	0	2,104
normal	google-NestHub-normal	-	166,627	0	1	1	0
normal	google-NestMini-normal-2	-	22,237	0	2	0	6,770
normal	google-NestMini-normal	-	150,036	0	1	2	1,796
normal	honeywell-C2WiFiHomeSecurityCamera-normal	-	47,437	0	0	1	0
normal	honeywell-W1WaterLeakAndFreezeDetector-normal	-	196	0	7	0	0
normal	leotec-AndroidTvBoxGCX2432-normal-2	-	200,974	0	2	1	0



(a) Tool running on Windows



(b) Tool running on Linux

Figure 7: Packet delays in two executions of NURSE with different OSes

hosted on dedicated subdomains or external domains which trigger new checks for each domain. If the number of new domains or CDN is too high and cause the delay to exceed the timeout value, this may cause an error on the browser, saying that the page is not available. However, when trying again after a few seconds, as the new domains have been processed, the page loads. The page loading may take longer than without monitoring due to the higher latency caused by the delay of our software.

We note that the ARP spoofing and traffic interception does not impact the network speed of the computer it runs on. However, some functionalities such as the blocklist checks of IP addresses could affect the connection by sending one request per contacted IP. That is why NURSE offers the possibility to disable these functionalities in the configuration page.

5 ETHICS AND PRIVACY CONSIDERATIONS

In this study, we leveraged a passively-collected datasets regarding infected IoT devices to test the accuracy of the tool. This is not regarded as human subject research by our IRB and thus out of scope. NURSE collects, stores, analyzes and displays the information only locally. Therefore, no data is sent to external servers and all the traffic data stays on the device the tool is running on. There is also no telemetry or similar features. However, our tool cannot run fully offline as it relies on some external services to identify the external IP, check if IP addresses are blacklisted, ease data plotting with JavaScript libraries, etc. These services use free and anonymous APIs, and no identifiable information is sent to these APIs. We also provide the possibility to disable some online functionalities such as blacklist checks if some users do not want to use this API.

A second privacy concern concerns people living in the same household and sharing the same network. A tool that performs traffic interception could be misused as a spying tool. NURSE user interface allows to add additional checks (e.g., introducing the MAC address of the devices that want to be monitored) to mitigate potential misuse.

6 LIMITATIONS

While NURSE could be extended to analyze encrypted protocol, in its initial design it only monitors unencrypted protocols. This is for one main reason: inspecting encrypted protocols, for example TLS, is doable in a MITM position as we have, but requires decrypting and re-encrypt traffic which requires a new certificate to be installed on the spoofed device. As this task requires complex operations to be performed on the device, users may not be allowed or competent enough to install such a certificate.

NURSE detection engine partially depends on third party reputation block lists. While these are not critical for the functioning of NURSE, it allows flagging single-packet alerts faster without inspecting other features. However, blocklists present their own limitations which are in turn inherited by NURSE.

NURSE might increase the latency of the network while it is monitoring it, which in turn might impact the usability of devices when monitored. However, the speed loss and connection issues are temporary, and in any case led to connection disruptions. Therefore, while it could affect some devices that rely on fast communications, we believe that IoT devices that only perform regular exchanges with servers will not be affected. We however do not recommend using NURSE on devices were the network availability or minimal network speeds are critical.

7 CONCLUSION

In this paper we have presented NURSE, a tool that enables end-users to detect whether any of their IoT devices are infected with malware. NURSE performs traffic interception to flag devices whose network behavior is associated with malicious activities. By leveraging a simple user interface and minimal setup process, NURSE allows non-savvy Internet users to monitor their own home networks.

NURSE is a multiplatform tool compatible whit Unix and Windows operating systems whose detection engine relies on a set of rule-based detection modules to trigger alerts. The rules that are used were designed observing actual malware and normal traffic from public datasets and malware traffic captures from a sandbox. We also extended our solution with a classifier of domains to help in identifying domains generated by domain generation algorithms used in malware. This classifier combined techniques from state-of-the-art DGA classifiers, but adapted the classification to work on a local machine and run with no need of external databases that would not be available on a home network.

We finally evaluated our solution in 83 different IoT network scenarios, focusing on two aspects: the botnet detection task, and the performance in home networks. Results showed that NURSE raised alerts in all scenarios where a malware-infected IoT device was present. However, some attacks that were not related to malware (e.g., sporadic unauthorized access) were not flagged by NURSE.

REFERENCES

[1] 2013. sshd_config(5) Linux man page. (04 2013). https://linux.die.net/man/5/sshd

[2] 2021. Shodan: the search engine for the internet of things. (2021). https://www.shodan.io/

[3] Sharad Agarwal, Pascal Oser, and Stefan Lueders. 2019. Detecting IoT Devices and How They Put Large Heterogeneous Networks at Security Risk. *Sensors* 19, 19 (2019). https://doi.org/10.3390/s19194107

[4] Alexa. 2021. Alexa Top Sites for Countries. (2021). https://www.alexa.com/topsites/countries

[5] Kamal Alieyan, Ammar ALmomani, Ahmad Manasrah, and Mohammed M Kadhum. 2017. A survey of botnet detection based on DNS. *Neural Computing and Applications* 28, 7 (2017), 1541–1558.

[6] Erdem Alparslan, Adem Karahoca, and Dilek Karahoca. 2012. BotNet detection: Enhancing analysis by using data mining techniques. *Advances in Data Mining Knowledge Discovery and Applications* (2012), 349.

[7] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE symposium on security and privacy (sp)*. IEEE, 1362–1380.

[8] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. 2019. Keeping the smart home private with smart (er) IoT traffic shaping. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 128–148.

[9] Sunny Behal, Amanpreet Singh Brar, and Krishan Kumar. 2010. Signature-based botnet detection and prevention. In *Proceedings of International Symposium on Computer Engineering and Technology*. 127–132.

[10] Philippe Biondi and the Scapy community. 2021. Scapy. (2021). https://scapy.net/

[11] Orçun Çetin, Carlos Ganán, Lisette Altena, Takahiro Kasama, Daisuke Inoue, Kazuki Tamiya, Ying Tie, Katsunari Yoshioka, and Michel van Eeten. 2019. Cleaning Up the Internet of Evil Things: Real-World Evidence on ISP and Consumer Efforts to Remove Mirai.. In *NDSS 2019*.

[12] Rohan Doshi, Noah Apthorpe, and Nick Feamster. 2018. Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 29–35.

[13] Chris Duckett. 2020. Chromium DNS hijacking detection accused of being around half of all root queries. (24 08 2020). https://www.zdnet.com/article/chromium-dns-hijacking-detection-accused-of-being-around-half-of-all-root-queries/

[14] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. 2009. A Survey of Botnet and Botnet Detection. In *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. 268–273. https://doi.org/10.1109/SECURWARE.2009.48

[15] S. García, M. Grill, J. Stiborek, and A. Zunino. 2014. An empirical comparison of botnet detection methods. *Computers & Security* 45 (2014), 100–123. https://doi.org/10.1016/j.cose.2014.05.011

[16] Google. 2021. Google SafeBrowsing API. (2021). https://developers.google.com/safe-browsing/v4/

[17] Guofei Gu, Junjie Zhang, and Wenke Lee. 2008. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the Network and Distributed System Security Symposium. (NDSS 2008)*.

[18] Hang Guo and John Heidemann. 2020. Detecting IoT Devices in the Internet. *IEEE/ACM Transactions on Networking* 28, 5 (2020), 2323–2336. https://doi.org/10.1109/TNET.2020.3009425

[19] Eszter Hargittai, Anne Marie Piper, and Meredith Ringel Morris. 2019. From internet access to internet skills: digital inequality among older adults. *Universal Access in the Information Society* 18, 4 (2019), 881–890.

[20] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. 2020. IoT Inspector. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 2 (6 2020), 1–21. https://doi.org/10.1145/3397333

[21] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. (May 2021). https://doi.org/10.17487/RFC9000

[22] M. O'Reirdan J. Livingood, N. Mody. 2012. *An Ethernet Address Resolution Protocol*. RFC 6561. RFC Editor. https://datatracker.ietf.org/doc/html/rfc6561

[23] Elmer Lastdrager, Cristian Hesselman, Jelte Jansen, and Marco Davids. 2020. Protecting home networks from insecure IoT devices. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–6.

[24] Knud Lasse Lueth. 2020. State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time. (19 11 2020). https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/

[25] Eman Maali, David Boyle, and Hamed Haddadi. 2020. Towards Identifying IoT Traffic Anomalies on the Home Gateway: Poster Abstract. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys '20)*. Association for Computing Machinery, New York, NY, USA, 735–736. https://doi.org/10.1145/3384419.3430414

[26] Majestic. 2021. Majestic million domain list. (2021). https://fr.majestic.com/reports/majestic-million

[27] Linda Markowsky and George Markowsky. 2015. Scanning for Vulnerable Devices in the Internet of Things. https://doi.org/10.1109/IDAACS.2015.7340779

[28] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W Felten, Prateek Mittal, and Arvind Narayanan. 2019. Watching you watch: The tracking ecosystem of over-the-top tv streaming devices. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 131–147.

[29] Netlab. 2021. Netlab OpenData Project. (30 03 2021). https://data.netlab.360.com/dga/

[30] OpenDNS. 2014. public domain lists. (6 11 2014). https://github.com/opendns/public-domain-lists

[31] Mockapetris P. 1987. *Domain names - Implementation and Specification*. RFC 1035. RFC Editor. https://datatracker.ietf.org/doc/html/rfc1035

[32] A Parmisano, S Garcia, and MJ Erquiaga. 2020. A Labeled Dataset with Malicious and Benign IoT Network Traffic. *Stratosphere Laboratory: Praha, Czech Republic* (2020).

[33] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. 2020. IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis. In *2020 IEEE European Symposium on Security and Privacy (EuroS P)*. 474–489. https://doi.org/10.1109/EuroSP48549.2020.00037

[34] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. 2020. IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 474–489.

[35] Imanol Rubio-Unanue, Iñigo Ortega-Martinez, Markel Baskaran, Eneko Bermejo, Antton Rodriguez, Igor Santos, Iñaki Garitano, and Urko Zurutuza. 2021. IoT security - network traffic under normal conditions. (31 03 2021). https://data.europa.eu/data/datasets?keywords=variot Mondragon Unibertsitatea: Data Analysis and Cybersecurity Research Group [producer]. European Data Portal [distributor].

[36] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J. Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. 2020. A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild. In *Proceedings of the ACM Internet Measurement Conference (IMC '20)*. Association for Computing Machinery, New York, NY, USA, 87–100. https://doi.org/10.1145/3419394.3423650

[37] Farsight Security. 2020. DNSDB. (2020). https://docs.dnsdb.info/

[38] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2018. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2018), 1745–1759.

[39] Spamhaus. 2021. The Spamhaus project. (2021). https://www.spamhaus.org/

[40] Spartan2194. 2019. Detecting SSH brute forcing with Zeek. (17 04 2019). https://holdmybeersecurity.com/2019/04/17/detecting-ssh-brute-forcing-with-zeek/

[41] Rui Tanabe, Tatsuya Tamai, Akira Fujita, Ryoichi Isawa, Katsunari Yoshioka, Tsutomu Matsumoto, Carlos Gañán, and Michel Van Eeten. 2020. Disposable botnets: examining the anatomy of iot botnet infrastructure. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 1–10.

[42] Te-k. 2021. Pysafebrowsing. (2021). https://pypi.org/project/pysafebrowsing/

[43] Sean Whalen. 2001. An introduction to arp spoofing. *Node99 [Online Document]* (2001).

[44] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. 2014. IoT security: ongoing challenges and research opportunities. In *2014 IEEE 7th international conference on service-oriented computing and applications*. IEEE, 230–234.