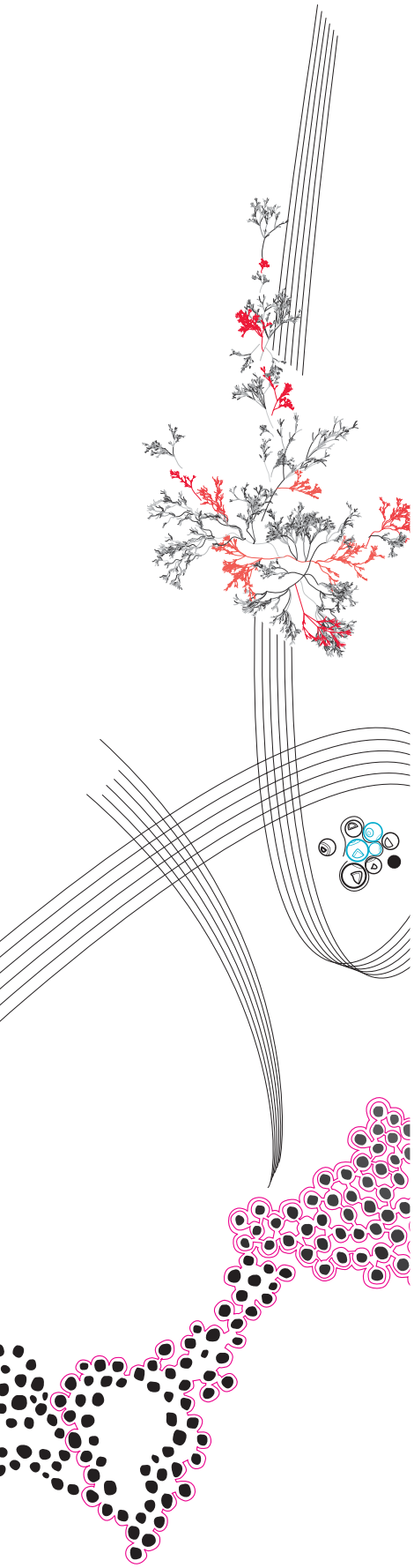

MSc Computer Science
Final Project

Threat Hunting and Analysis for IoT Malware

Richard Faragó

Supervisor: dr.ir. Andrea Continella (UT) &
Bertalan Borsos (ELTE) & Daniel dos Santos (Forescout)

Committee member: Roland van Rijswijk-Deij (UT)

August, 2024

UNIVERSITY OF TWENTE.

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

Abstract

The surge in the adoption of Internet of Things (IoT) devices has been substantial and caused a rise in malware attacks targeting them. This research delves into the emerging threat landscape of the IoT technology, specifically seeking to enhance the understanding of malware behaviors and attack vectors aimed at devices with embedded operating systems. Furthermore, we outline a comprehensive methodology for collecting, analyzing and visualizing IoT malware data, while also capturing new malware samples through a novel approach. In the end, we contribute to the development of an up-to-date IoT Malware Knowledge Base.

Keywords: Internet of Things (IoT), IoT malware, malware catalog, malware dataset, malware characteristics, malware collection, malware analysis, VirusTotal, metadata, Intelligence Search, Yara, Livehunt, Retrohunt, behavioral indicators, static analysis

Acknowledgements

This research would not have been possible without the invaluable support and contributions of many people and organizations.

First and foremost, we would like to express our sincere gratitude to the Vedere Research Labs of Forescout Technologies for providing us with the opportunity to conduct this research during our internship at the company. We are particularly grateful for the chance to delve into the fascinating field of threat hunting, particularly in the growing area of the IoT threat landscape.

We want to thank our mentor, Simon Guiot, and our supervisors, Daniel dos Santos and Alessandro Manzi, for their amazing guidance throughout this journey. Their trust in us let us independently explore and develop our research workflow and plan, which helped us feel empowered and motivated to take on this challenge and provide meaningful results to the community. Whenever we faced obstacles or needed direction, they were always available to help us, which was crucial to our progress.

We are also incredibly thankful for the supportive and knowledgeable subject matter experts on the research team. Their expertise in various domains proved invaluable, and we are grateful for their willingness to share their knowledge and insights, especially on topics outside our area of familiarity.

This research greatly benefited from access to VirusTotal's premium services like Retrohunt, Livehunt, and Intelligence Search which Vedere Labs provided access to. We are grateful for the chance to use these tools. Additionally, the open-source bincapz tool was also a great help during our behavioral extraction process, and we're thankful for the team that developed it.

Beyond the company, we would like to thank Malpedia for generously providing us with a student account. Access to their advanced features, providing us with IoCs and YARA rules for malware families, significantly enriched our research.

Finally, we owe a big debt of gratitude to our supervisors at the University of Twente (UT) and Eötvös Loránd University (ELTE), Andrea Continella and Bertalan Borsos, for their invaluable feedback and unwavering support throughout this project. Their insights and guidance were crucial in shaping this research.

Thanks to everyone who contributed to this work.

Contents

Chapter 1

Introduction

In recent years, smart home appliances have gained widespread adoption and several industries have seen a significant growth in the use of Internet of Things (IoT) devices thanks to their capabilities to enable automation, connectivity and ease of use [1]. Substantially, IoT devices refine both household activities [2] and industrial processes [3].

IoT devices are everyday electronic objects that use several technologies to communicate with one another and are connected to the Internet [4]. IoT devices encompass edge devices [5] and consumer equipment [6] as well, and they are used in diverse applications like object identification, logistics, maintenance, monitoring and controlling environments such as smart homes, smart cities and industrial sectors. IoT devices are designed to collect and exchange data, while being integrated with Operational Technology (OT) systems [7].

The proliferation of IoT devices has led to a concerning rise in targeted malware [8, 9]. The infamous 2016 Mirai botnet attack [10], which leveraged compromised IoT devices for large-scale denial-of-service attacks, illustrated the potential consequences of vulnerabilities found in IoT devices [11]. The release of Mirai's source code [12] further fueled a surge in the development and deployment of IoT malware. To effectively counter this evolving threat landscape, understanding current malware tactics, techniques, and procedures [13] is crucial. This understanding relies on continuous collection and analysis of threat intelligence [14] from sources like online malware repositories, vendor reports and bulletins, malware forums, blog posts and technical reports by security researchers.

Recent surveys [15, 16] have attempted to catalog the growing IoT malware landscape, but these efforts have been constrained by manual data collection methods that assigned malware characteristics, such as purpose, manually. In contrast, our approach, although initially manual, shifts to an automatic approach based on behavioral indicators once a sufficient amount of data has been analyzed. Additionally, the rapid emergence of new malware variants has also impacted the prior studies, as discussed further in Section 3.2. As a result, previous works are outdated, covering malware families and variants only up to a certain year. Additionally, the number of variants collected by the authors of the referenced papers is relatively small compared to the increasing prevalence of IoT malware that our work has uncovered. Our research has identified nearly three times as many IoT malware variants as the largest prior study.

Building upon previous research, our study seeks to broaden the understanding of the evolving IoT malware threat landscape. We aim to analyze key malware attributes and uncover relationships between malware characteristics, with the goal of developing a novel approach for effective threat hunting and identification of emerging IoT malware samples on online repositories (Chapter 3).

To accomplish our objectives, we follow three main steps. First, we build an IoT Malware Knowledge Base and create two distinct malware datasets. One dataset focuses on specific IoT malware samples with a singular type and purpose, while the other consists of a more

comprehensive and generic dataset containing malware samples with varied types and purposes. Second, we analyze the malware behaviors across these datasets using static analysis tools, and extract indicators that correspond to the observed behaviors to develop detection rules. Finally, we aim to identify emerging IoT threats through deploying these detection rules on popular online malware repositories, and further analyzing the malware samples that match the criteria. More details on our methodology are provided in Chapter 4.

By applying our methodology, we successfully compiled an up-to-date catalog of IoT malware and also identified new variants using a novel technique based on behavioral indicators. Chapter 6 details our findings and validates the effectiveness of our approach in collecting and analyzing IoT malware.

In summary, this research makes the following key contributions to the field of IoT malware:

- A more comprehensive and up-to-date IoT Malware Knowledge Base, covering malware families and variants up to early 2024, in contrast with previous works that only went up to 2018 and 2022. This new database contains 192 variants, a substantial increase over the 60 and 77 variants in prior versions.

- An in-depth analysis of valuable malware attributes gathered during the construction of the IoT Malware Knowledge Base. While these attributes were identified before, their relationships, such as how unique vulnerabilities or sources can link different malware families, have not been thoroughly explored until now.

- The construction of two IoT malware datasets for further analysis - one containing specific IoT malware samples, such as wiper malware, and another comprising a more generic and diverse set of malware samples with varying types and capabilities.

- The development of a novel technique for effective threat hunting and identification of emerging IoT malware on online repositories by discovering new variants as well as new samples of known malware families.

Chapter 2

Background

This section establishes the necessary context and background knowledge to comprehend the focus of this thesis. It systematically organizes relevant information into subsections that explore the nature and features of malware, the cybersecurity landscape and problems in the IoT domain, how to obtain malware samples, the process of analyzing malware, the essential platforms and tools used, and finally, the challenges involved in malware analysis.

2.1 Malware

In the context of IoT devices, we refer to malware as Executable and Linkable Format (ELF) files, which is the predominant file format for binaries used in Unix-based embedded systems like those found in IoT devices [17]. Throughout the subsequent sections, the majority of the malware referred to as IoT malware will be of the ELF file type. We also acknowledge that malware targeting IoT device ecosystems can utilize other file formats, such as shell scripts, in addition to ELF binaries. However, these alternative file types tend to be less sophisticated compared to ELF binaries, which can present additional challenges for malware analysts during the analysis process, as discussed further in Section 2.5.

While IoT malware often exhibits distinct characteristics, such as command-and-control communication for managing botnets and including commands for various botnet attack types, which differ from malware targeting traditional computing platforms like desktops or servers. However, IoT malware is not restricted to targeting only IoT devices [18]; it can also infect a range of other platforms [19], including cloud services and Docker instances.

IoT malware are capable of various malicious behaviors, allowing them to be classified into distinct categories based on their intended functionality and purpose. Some common types of IoT malware include botnets [20], worms [21], droppers [22], backdoors [23], and wipers [24]. IoT malware can engage in a diverse range of activities, such as data exfiltration, Distributed Denial-of-Service (DDoS) attacks, cryptocurrency mining, white-hat activities, propagation, destruction, and the facilitation of proxy services [16].

2.2 Cyber Attacks and Security Issues

IoT devices are often targeted due to their Internet exposure and widespread use across various industries. These industries, including telecommunications (e.g., modems, routers), surveillance and security (e.g., IP cameras, DVRs), and entertainment (e.g., gaming consoles, smart TVs), feature numerous devices vulnerable to cyberattacks [25]. IoT malware can be implanted in these devices, leading to various cyber threats. Recent examples include the Coathanger Remote

Access Trojan (RAT) used to target the Dutch Ministry of Defence [26] and the AcidPour wiper deployed to disrupt Ukrainian telecommunication networks [27].

These attacks are often successful due to several critical security issues arising from the diversity of devices, their applications, and their lack of security features [17]. Exploiting insecure default configurations and poor user security awareness (e.g., weak passwords, open ports) are common attack strategies. Additionally, resource constraints often limit the use of strong encryption, leading to insecure communication. Unsecured firmware update mechanisms and the use of End-of-Life products with unpatched vulnerabilities further increase these risks.

2.3 Malware Sample Collection and Analysis

Following a cyber attack or security incident, incident response teams rapidly collect and analyze data to understand the threat. However, the team requires the prior implementation of defensive measures, which may encompass either automated or manual approaches.

The automated approach involves using security tools and platforms to capture and catalog potentially malicious artifacts. One effective technique is deploying honeypots - decoy systems designed to lure and trap attackers [28]. These honeypots provide valuable insights into the attackers' methods and intentions by monitoring their interactions with the system, such as network traffic, running processes, and file system changes. For example, an emulated shell that appears to be a regular breached system can deceive attackers into interacting with the controlled environment, allowing honeypots to log their activities and preserve the artifacts they use to compromise the system. This approach offers invaluable insights into the attackers' techniques and intentions.

The manual approach [15], on the other hand, relies on security analysts manually searching the Internet for samples using various online resources. A platform such as Malpedia[1] curate these resources, offering a centralized source of information on the latest trends and developments in IoT malware [29, 30].

Researchers often analyze malicious activities detected on honeypots or reported in incidents, and share their findings through blog posts, articles, or technical reports. These publications often include hashes or links to third-party malware repositories (e.g., MalShare, MalwareBazaar, VirusTotal, VirusShare) containing the relevant binary samples, allowing other analysts to download and use those artifacts in their research as well [17, 30].

After collecting samples, malware analysis is a key part of understanding the threats. By studying the behavior, functionality, and characteristics of the malware, researchers can gain insights how they work and what risks they pose to the device itself and the networks they are part of [17, 31].

One approach that involves studying the behavior of malware at runtime is called dynamic analysis. The execution of the malware happens in a controlled environment which allows the malware to run seamlessly by simulating the targeted machine. There are several sandboxed environments [17] based on supported file formats, cpu architectures and other features. One particularly notable tool is Drakvuf, which leverages hardware virtualization and the Xen hypervisor to facilitate in-depth, reliable malware studies [31, 32].

On the other hand, static analysis facilitates the examination of the malicious file's content and its structural composition without executing it. Some of the techniques used consist of function call and string extraction, opcode, string and file header analysis, and gray-scale image comparison [17, 33].

State-of-the-art solutions involving both static and dynamic analysis are presented in greater detail in Chapter 7.

[1] https://malpedia.caad.fkie.fraunhofer.de/

2.4 Platforms and Tools

2.4.1 YARA

YARA [34] is an effective rule-based text matching tool. YARA rules are essentially descriptions of malware characteristics written in a human-readable format. These rules can include text strings, binary patterns, regular expressions, and even more complex logic achieved through the use of modules. YARA offers a variety of modules to enhance its analytical capabilities. For instance, the `elf` [35] module allows for the inspection of ELF file structures, commonly found in Unix-based IoT devices. Similarly, the `math` [36] module enables mathematical operations within YARA rules, useful for calculating entropy [37] or performing other arithmetic checks on the binary data. Additionally, modules like `vt` [38] can integrate with external services like VirusTotal, allowing YARA rules to leverage threat intelligence data during analysis on those platforms.

2.4.2 bincapz

Bincapz [39] is another handy tool, which focuses on extracting capabilities from binaries, essentially revealing what actions a particular binary is capable of performing based on open-source YARA rules. This information is crucial for understanding the potential impact of a malware sample.

2.4.3 VirusTotal

Finally, VirusTotal[2], an online malware analysis platform, offers an extensive repository of previously analyzed samples, enabling researchers to search for and identify new variants or related malware. VirusTotal's intelligence search also allows querying its vast database using various metadata, such as hashes, IP addresses, file names, or even the content of the samples themselves. Additionally, VirusTotal's services like Retrohunt and Livehunt offer the ability to analyze historical data and gain real-time insights into emerging threats and malware trends. VirusTotal also offers a command-line interface (CLI)[3] tool that allows researchers to easily integrate and automate the use of VirusTotal's services within their own analysis workflows by interfacing with its API.

2.4.4 AVClass

By leveraging a tool called AVClass[4], we can analyze VirusTotal reports to extract the malware names or labels assigned by various antivirus engines. This can provide valuable insight into the broader malware landscape represented in the dataset, as demonstrated by the prior research conducted by Emanuele Cozzi and his colleagues [40]. While they did not rely on AVClass for primary sample labeling, they used it to gain insights into the potential family affiliations of the collected samples. This approach likely offered useful context for their further analysis of code relationships and evolutionary patterns within the dataset.

[2] https://docs.virustotal.com/
[3] https://virustotal.github.io/vt-cli/
[4] https://github.com/malicialab/avclass

2.5 Challenges

Malware creators use a variety of techniques during malware development to enhance portability, evade detection, and hinder analysis. One such technique is static linking, which makes binaries more portable by including all necessary dependencies, while simultaneously making them harder to reverse engineer as the used libraries may be unknown. Additionally, obfuscation [37] techniques can render static analysis less effective by masking the true functionality of the code. Similarly, packers like Vanilla UPX, Custom UPX variants, and Mumblehard are used for the same purpose of making the malware more challenging to analyze [17]. Furthermore, manipulating ELF headers can mislead or even crash static analysis utilities, as the intention is to make files appear anomalous or invalid. Other persistence techniques, such as subsystem initialization, time-based execution, file infection and replacement, and user file alteration, allow malware to survive reboots and remain hidden. Choosing a common service name, an empty, or random-like name can also help the malware evade security tools [18].

Evasion techniques enable malware to detect debuggers and virtual environments, leading to behavioral changes like delayed execution, termination, or even the radical deletion of the entire file system [31]. Methods such as sandbox detection, which relies on comparing strings extracted from the targeted system, and process enumeration based on checking for specific running processes, can be used. Stalling code, which refers to sleep-related functions, can be used to delay execution. Anti-execution checks whether the malware itself was renamed before execution, and anti-debugging features identify if the process is already attached to a debugging instance - both of which are signs of malware analysis that can lead to the unexpected termination of the malware.

In Chapter 8, we further explore these challenges in the context of the limitations of this research.

Chapter 3

Motivation

This chapter outlines the problem statement and research questions, discusses existing knowledge bases, and presents the goals for this research.

3.1 Problem Statement

To understand IoT malware and develop anti-malware strategies, security researchers and analysts require rich datasets with detailed characteristics of malware families. While current IoT malware datasets exist, concerns may arise about their completeness and potential outdatedness given the limited number of such datasets currently available. Consequently, this thesis analyzes this issue, gathers comprehensive and current data on IoT malware families and their characteristics, and compares the new dataset's novelty with previous ones. Furthermore, this study also assesses the effectiveness of the collected characteristics in identifying existing IoT malware samples on online malware repositories. Additionally, the increasing availability of metadata from online malware repositories opened up a new opportunity to hunt for new and emerging IoT malware variants more effectively. However, this method has not been researched or evaluated before, indicating a need for further investigation within the scope of this research. In order to address these gaps, we answer the following main questions:

- To what extent are current IoT malware datasets comprehensive and up-to-date with respect to the existing variants and characteristics of malware families?
- What unique signatures or characteristics are instrumental in reliably identifying known IoT malware families?
- To what extent can metadata from online malware repositories be deemed reliable for discovering new samples of IoT malware?

Answering these questions helps us assess the current landscape of IoT malware and a new approach to threat hunting, ultimately contributing to more robust security practices in the interconnected world.

3.2 Existing Knowledge Bases

As IoT malware have gained significant attention in recent years, there have been studies focused on surveys regarding IoT malware, such as Costin and Zaddach's work identified 60 IoT malware variants dating back from 2005 to 2018 [15], and the more recent study by Victor et al. collected 77 between 2008 and 2022 [16]. Even though they gathered an extensive list of variants, they used different methods to accomplish it.

In the former, the authors address the need for a comprehensive analysis of IoT malware. They acknowledge the limitations of existing research, which often relies on scattered blog posts, technical reports, or studies focused on specific malware strains. These resources, they argue, are prone to errors, omissions, and limited perspectives, hindering efforts to systematically analyze, track, and defend against IoT malware. To address this gap, the authors undertook a largely manual process of collecting, archiving, and cross-validating over 637 unique resources related to IoT malware families. This approach allowed them to create a more comprehensive and reliable dataset than previously available. They then used this data to establish timelines of events related to each malware family and identify important insights and statistics. The authors acknowledge the challenges of their manual approach, such as the potential for errors or inconsistencies in source material, the limited time availability of resources, and the difficulty of maintaining an up-to-date dataset. However, their work represents a significant step towards a more systematic understanding of the evolving landscape of IoT malware, and served as a starting point for our IoT Malware Knowldege Base, while we addressed some of the difficulties they encountered by using specific tools and techniques further presented in Chapter 4 and 5.

The latter paper opted for addressing the lack of a unified understanding of IoT malware, which hinders effective security research. The authors tackle this by presenting a novel, attribute-based taxonomy of IoT malware. This taxonomy comprises 100 attributes categorized into ten key aspects of IoT malware, including attack types, victim devices, programming languages, and more. Their state-of-the-art contribution is twofold: a comprehensive taxonomy - instead of relying on broad classifications, they meticulously break down IoT malware into granular attributes. This allows for a more precise and nuanced understanding of malware characteristics, enabling researchers to identify patterns, relationships, and potential vulnerabilities more effectively; and large-scale mapping - the authors apply their taxonomy to 77 real-world IoT malware families identified between 2008 and 2022. This mapping demonstrates the taxonomy's practicality and provides valuable insights into how common different malware features are and how they have changed over time. By offering this detailed taxonomy and real-world mapping, the authors provide a valuable method for systematically analyzing existing malware. However, the dataset still lacks the full spectrum of IoT malware as new variants have been constantly emerging.

3.3 Goals

The goal of this thesis is to build upon these existing efforts and further advance the understanding of IoT malware by addressing the limitations of the current datasets, as well as exploring new techniques for threat hunting and analysis.

Specifically, this research develops a more extensive IoT Malware Knowledge Base with focus on up-to-dateness by incorporating new variants and families, however, still largely relying on manual efforts. Furthermore, this work explores the potential of enhancing static analysis with behavioral indicators derived from the collected malware characteristics, aiming to improve the accuracy of threat hunting. While the use of metadata provided by third-party platforms (e.g. VirusTotal) for novel malware discovery proved less fruitful than initially anticipated, its potential evolution is worth further investigation in future research. By pursuing these goals, this dissertation strives to contribute to a deeper and more well-rounded understanding of the evolving threat landscape of IoT devices.

Chapter 4

Approach

In order to achieve the goals outlined in the previous chapter, we divided the research into three key steps:

1. Expand the IoT Malware Knowledge Base
2. Enhance static analysis with behavioral indicators
3. Explore novel malware discovery techniques

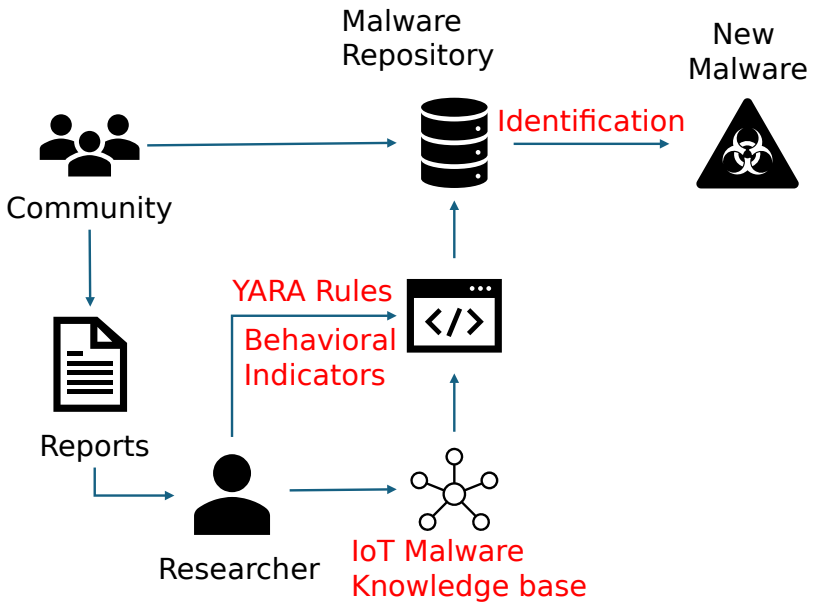

FIGURE 4.1: The diagram illustrates the components of the research process. The red labels mark our contribution.

The diagram in Figure 4.1 outlines the three key steps of this research. It illustrates the various entities involved, including the `Community` that provides malware samples to the online `Malware Repository`, and the `Reports` containing information about IoT malware and related

incidents. The first step is depicted in the diagram by an arrow pointing from the `Researcher` to the `IoT Malware Knowledge Base`, representing our efforts to create a new, comprehensive knowledge base on IoT malware by examining various reports. The second step is shown by two arrows, one from the `Researcher` and one from the `IoT Malware Knowledge Base`, pointing towards the `YARA Rules`, indicating the process of incorporating behavioral indicators into the YARA rules based on the analysis of samples from the datasets built using the knowledge base. The third step involves using the created YARA rules to identify new malware samples from the repository platform represented by the arrow between the `Malware Repository` and `New Malware`.

4.1 Expand the IoT Malware Knowledge Base

To expand the IoT Malware Knowledge Base, we used mainly manual approaches. We started by conducting a thorough review of existing literature, reports, and online resources to identify new IoT malware families and their different variants that were not included in previous surveys. This involved searching through security blogs, vendor reports, other online publications, and open-source platforms that often served as a catalog of useful online resources. We then cross-validated the collected information from multiple sources to ensure accuracy and completeness.

For each new malware variant we noted down detailed attributes, such as type, purpose, variant, language, timeline, infected countries, country of origin, attribution, targeted devices and initial access techniques. Each of the characteristics have their own specific description and predefined values. The type (e.g., worm, dropper, backdoor, wiper) property describes the main capabilities of the malware. The purpose attribute refers to the intentions (e.g., Distributed Denial-of-Service (DDoS), espionage, exfiltration, cryptomining, white hat, propagation, destruction, proxy server) of the threat actor deploying the malware. The variant field identifies which previously discovered malware the current sample is similar to, while the language field describes the programming languages used to develop the malware. The timeline depicts the series of dates based on the first appearance, the emergence of new variants, the spotted cyber attack involvements and in most cases also the discontinuation of its activities. The attacked countries specify where targeted infrastructures were located or the list of the countries where infected devices lay. Infected devices refers to a list of compromised components that can be used as part of a botnet attack [20] or to operate as a proxy service. The attribution property indicates the alleged threat actors such as Advanced Persistent Threats (APT) and cyber criminal groups behind the deployment of attacks and/or creation of the malicious tools. The country of origin tag is self explanatory and sometimes derives from the fact that the creators publicly claim it, although in most cases there are leftovers in the source or reverse engineered code that hint at its origins. The targeted devices attribute is a list of vendors and models. The last characteristic, corresponding to initial access tactics (based on the common knowledge base of the MITRE ATT&CK Framework [41]), indicates how attackers compromised the devices. This field includes the Exploit Public-Facing Application technique (T1190[1]) using exposed unsecured protocols (e.g., http, telnet, ssh), and the Valid Accounts (T1078[2]) method based on vulnerable credentials. In addition, the initial access property lists Common Vulnerabilities and Exposures (CVE) that were exploited in the targeted devices to achieve access to them. This list of CVEs can be used to determine what are the targeted devices by using the database provided by the National Institute of Standards and Technology (NIST)[3].

While collecting the list of CVEs for each variant, we encountered a challenge. Some of the vulnerabilities identified in online resources were not officially assigned CVE identifiers.

[1] https://attack.mitre.org/techniques/T1190/
[2] https://attack.mitre.org/techniques/T1078/
[3] https://nvd.nist.gov/vuln/search

Instead, they were described loosely using a combination of the affected device name and the vulnerability type (e.g., Netgear unauthenticated remote code execution, Eir Wan Side remote command injection). This issue rose from either incomplete/missing information or inconsistent linkage between the exploit database and the vulnerability identifiers. To address this challenge, we performed additional research to locate the corresponding vulnerability identifiers for each vulnerability, wherever possible, and link them to the associated exploits. This process is further discussed in Section 5.1.

We believe that contributing to these websites can lead to an overall better vulnerability management and provide richer data feeds for institutions and organizations that use such sources in their security practices.

4.2 Enhance Static Analysis with Behavioral Indicators

Building upon the expanded IoT Malware Knowledge Base, our research then focused on enhancing static analysis capabilities because online malware repositories enable the deployment of detection rules relying on static analysis. Our goal was to develop a novel approach leveraging behavioral indicators to enhance the existing static analysis techniques. To this end, we incorporated behavioral indicators derived from the collected malware characteristics, such as the type and purpose. Our intuition was that behavioral indicators could potentially uncover patterns and anomalies, specific to malware characteristics, that would complement and strengthen the existing static analysis approaches.

We first focused our analysis on IoT malware with the specific type of `wiper` functionality and the specific purpose of `destruction`. We chose this due to the limited research attention it has received, as well as the recent high-profile destructive malware incidents involving embedded devices in Ukraine [27].

The next step involved collecting malware samples using resources found in our IoT Malware Knowledge Base, and subjecting them to further static analysis to identify specific behavioral indicators. We leveraged a combination of automated tools and manual analysis to gather these indicators associated to each malware variant, in some cases based on multiple samples for the same variant. During this stage, we not only collected them, but also evaluated their effectiveness in detecting the corresponding behavior across other samples. Where adjustments were necessary, we optimized the indicators accordingly.

The specific behavioral indicators we identified for the wiper category included: attempts to delete data, attempts to write junk data on specific device paths and attempts to either shut down or reboot the devices. After having developed and validated these indicators, we then incorporated them into rule-based malware detection tooling. The rule conditions based on the indicators were aimed to find potential new IoT malware samples for further investigation.

In continuation of this approach, we explored the potential to develop similar rule, however, based on a wider range of behavioral indicators, not just those specific to the wiper malware category. We gathered samples of IoT malware previously deployed by APT groups, and subjected them to the same process of static analysis and extraction of behavioral indicators. The resulting indicators covered a broader set of device manipulations, such as attempts to scan the local network to spread further from the infected device, attempts to communicate with command and control servers, attempts to use cryptographic components and evasion techniques to hide their true intentions, as well as attempts to drop files and execute programs. These more generalized behavioral indicators were then also incorporated into our detection tooling, enabling coverage for a wider range of IoT malware.

Further details on the use of these rules are provided in the following section, and information about the implementation and results is covered in later chapters.

4.3 Explore Novel Malware Discovery Technique

The final key step explored the potential of using the extracted malware characteristics to improve threat hunting. We investigated how different services, such as VirusTotal's Intelligence Search and Retrohunt functions, could be used to discover previously unknown IoT malware samples by leveraging the information gathered through our static analysis: the identified behavioral indicators and the created rules based on them. Additionally, we explored VirusTotal's Livehunt ruleset functionality to capture, in real-time, new or reanalyzed samples of IoT malware detected by our rules.

The VirusTotal Intelligence Search allowed us to query their extensive database focusing only on specific behavioral indicator patterns (not the entire rule) we had defined, this could be related to the content of the binary file which can be searched for in the form of a specific strings or bytes sequences, or regular expression patterns. Beside the content-based search, we could also leverage the various metadata properties provided by the tools integrated into VirusTotal's analysis pipelines. This includes information from `Detect It Easy (DIE)`[4] about the file's compilation and software components, `Exiftool`[5] insights into the hardware-specific characteristics, and third-party antivirus classification models that identify malicious files and assign malware family and type tags, as well as other relevant attributes.

The search functionality proved to be quite resource-intensive, causing us to encounter limitations quickly. More details on this constraint are provided in Chapter 8.

In contrast to the Intelligence Search, VirusTotal's Retrohunt capability allowed us to conduct retroactive searches on their entire corpus of scanned files, however, this time based on entire rule sets instead of individual indicators. Even though, we could have created rules with only few indicators, but we would have encountered other limitations related to the monthly quota of permitted Retrohunt jobs. As a result, we opted to run the complete rule, combining all the indicators developed through the static analysis process.

This approach involved submitting our customized rules to the VirusTotal service. After the processing was completed, we retrieved all the matched samples for further analysis. We were able to obtain the malware samples themselves as well as the various metadata provided by VirusTotal's analysis tools. Based on the results from the initial versions of the YARA rules, we identified the need to refine and optimize the rules, and then rerun the retroactive scanning to improve the accuracy of the indicator logic. After several iterations, we also deployed the refined rule on VirusTotal's Livehunt feature in addition to Retrohunt, allowing it to run for extended periods to assess the effectiveness of the optimizations. Once the rules proved sufficiently reliable, we conducted another Retrohunt job and maintained the Livehunt rule with optimizations for a longer duration.

The samples obtained through the VirusTotal-based techniques were then further analyzed, including automated AV classification clustering for the samples identified as malicious by AV engines, as well as manual investigation for the undetected samples where the numbers were manageable by us. The manual analysis revealed new IoT malware variants belonging to known families.

The implementation of these steps, the accuracy of the rules and the results obtained are elaborated on in the upcoming chapters.

[4] https://github.com/horsicq/Detect-It-Easy
[5] https://github.com/exiftool/exiftool

Chapter 5

Implementation Details

In this chapter we provide the implementation details of the approaches described earlier. We delve into the specific techniques and tools utilized to achieve the objectives. Both manual and automated steps are covered, including the challenges encountered and how they were addressed.

5.1 Creating the IoT Malware Knowledge Base

As we discussed in the previous chapter, the first step was to establish a knowledge base of IoT malware variants and families. Figure 5.1 provides an overview of the workflow involved in creating this knowledge base.

The process of building the IoT Malware Knowledge Base began with Internet research, exploring recent news about compromised IoT devices, newly discovered malware strains, and emerging threat groups targeting IoT infrastructure. After identifying multiple IoT malware

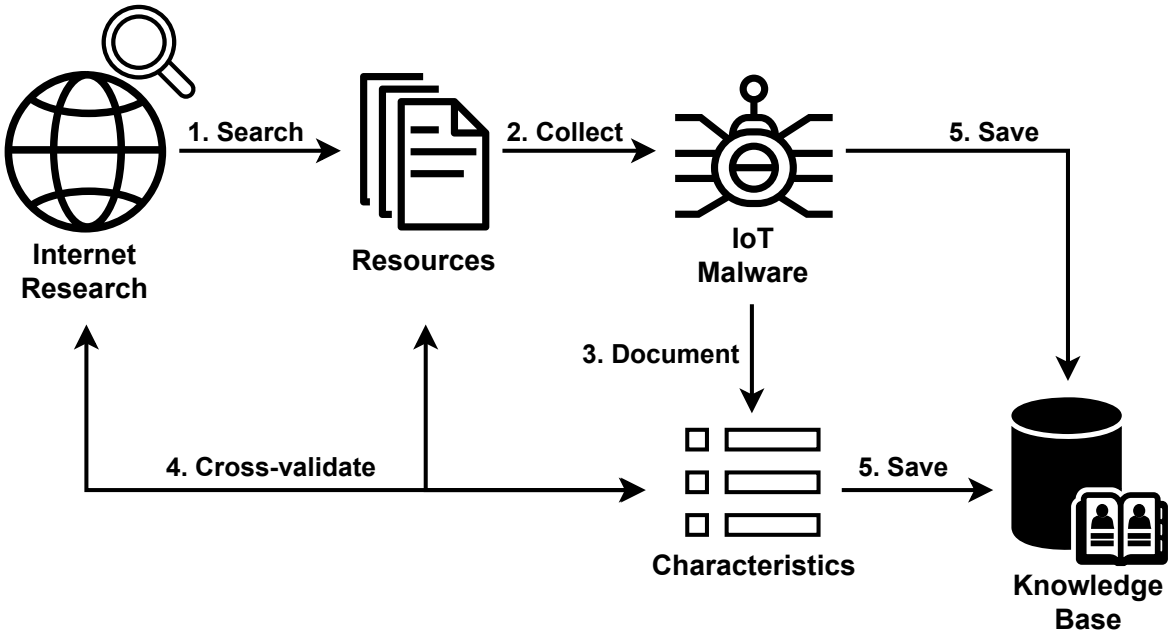

FIGURE 5.1: The diagram illustrates the key steps of the IoT Malware Knowledge Base creation, from internet research to saving the characteristics of malware variants into the knowledge base.

variants, we recognized common characteristics that could be documented for each one. We then systematically searched for and noted down the same attributes for each variant, cross-validating the details across multiple sources to ensure accuracy. These characteristics were previously outlined in Section 4.1.

We used a wiki platform, to collaboratively document and organize the growing knowledge base of IoT malware variants. As we identified more sophisticated malware deployed by nation-state threat groups, also known as APTs, we decided to divide the catalog into two separate lists - one for APT-deployed IoT malware [42] and another for cyber-criminal-deployed malware [43, 44]. There were instances where these categories overlapped, such as the MooBot botnet, which was initially developed by cyber-criminals but later utilized by the APT28 group [45]. This division provided a clearer understanding of the commonalities and differences between the two categories - cyber-criminals typically deploy malware targeting a large number of victims and use multiple versions, while APTs often have a more targeted approach towards specific region or institutions.

As we gathered information on individual IoT malware variants, the size of our documented knowledge base grew exponentially. This was because we often found additional malware names that were either similar in characteristics or connected in terms of common developers, authors, or distribution methods. These connections were frequently revealed through cross-references and links between research articles and malware analysis blogs.

Additionally, we discovered an online platform called Malpedia that aggregates and curates a comprehensive collection of malware resources. This platform categorizes each malware sample based on its target operating system, using prefixes such as `win` for Windows, `elf` for Unix, `apk` for Android, and `osx` for macOS. The platform also provided an inventory feed service, which allowed us to regularly obtain updates on the latest indexed malware and related news. We utilized this resource to enhance our knowledge base with additional IoT malware, carefully filtering for those with the Unix-specific prefix and verifying their relevance to the IoT threat landscape.

This platform not only lists resources and information on various malware families, but also provides details on the threat actors associated with each malware family, when available. This allowed us to integrate the newly discovered IoT malware variants into the appropriate section of our divided knowledge base. On top of this, we contacted the creators of the Malpedia platform to request an account that would provide access to additional functionalities, such as the ability to obtain hashes of samples related to the malware variants. This proved useful in a later stage of the research when we needed to collect samples for static analysis and extraction of behavioral indicators. More details on this will be provided in the following section.

One of the challenges we faced, which we did not yet mention in the previous chapters, was the temporary availability of resources. Certain websites or materials may have only been available for a particular time period until they either got removed, restricted, or the website became completely unavailable. To address this issue, we used the Internet Archive Wayback Machine[1], a platform that preserves and provides access to archived versions of websites, allowing us to retrieve the necessary content.

Another challenge was the lack of complete information about vulnerabilities identified in online resources, as some lacked explicit references to associated exploits or CVE identifications. However, as we discussed in Chapter 4, Section 1, we were still able to piece together relevant data by using Google Dorking [46], which involved leveraging the `site` and `intext` search query tags to filter Google results based on specific domains and page content, respectively. Additionally, we searched the NIST's vulnerability database (NVD) using keywords related to vendor, product, and software versions.

There were several cases where these methods helped uncover relevant details about IoT

[1] https://web.archive.org/

vulnerabilities and their exploitation. For instance, if the resource provided the exploit itself or a reference to the proof of concept (PoC), but did not include the corresponding CVE identifier. In such cases, we could search for specific elements of the code, such as the HTTP request method, URL path, headers, and payload, to identify the relevant CVE. When the exploit was missing from the resource, but it offered information about the exploit's name, type, and affected products, we were able to find the exploit through NVD searches.

Additionally, there were instances where we lacked sufficient information to definitively identify the corresponding CVE, or we found alternative identifiers instead of CVEs, either because the CVEs did not exist or their association was unclear to us. To supplement the CVE information, we utilized other vulnerability databases such as Exploit-DB[2], Packet Storm Security[3], Cybersecurity Help[4], and Vulners[5]. The Vulners platform aggregates vulnerability information from multiple sources, such as Exploit-DB and PacketStorm, allowing us to search for vulnerabilities across various databases through a single resource.

With the knowledge base of IoT malware variants largely compiled, we recognized that the data was in a raw format that made it difficult to perform further analysis and correlation. To address this, we decided to export the wiki page containing the information and parse the data using a Python script to generate a JSON-formatted file that could serve as input for more in-depth analysis. The representation of one entry of the knowledge base is shown in Appendix A.

With the IoT Malware Knowledge Base converted to a JSON format, we could conduct various analytical operations. This included identifying shared indicators, uncovering connections between malware families, and generating charts based on malware timelines or graphs depicting commonalities across variants.

To enable the data analytical operations mentioned earlier, we created an index on the ElasticSearch[6] analytics platform, which served as a database and provided an efficient way to query the extensive collected information. To present the data in a more accessible and visually appealing format, such as through the creation of dashboards and charts, we utilized the Kibana[7] visualization plugin. Additionally, using the previously constructed JSON format, we were able to generate the timeline chart and create multiple graphs that illustrate the relationships between different malware families. This was achieved through the use of Python scripts to preprocess the data, along with specific libraries such as Matplotlib[8] and Jaal[9].

[2] https://www.exploit-db.com/
[3] https://packetstormsecurity.com/
[4] https://www.cybersecurity-help.cz/
[5] https://vulners.com/
[6] https://www.elastic.co/elasticsearch
[7] https://www.elastic.co/kibana
[8] https://matplotlib.org/
[9] https://mohitmayank.com/jaal/

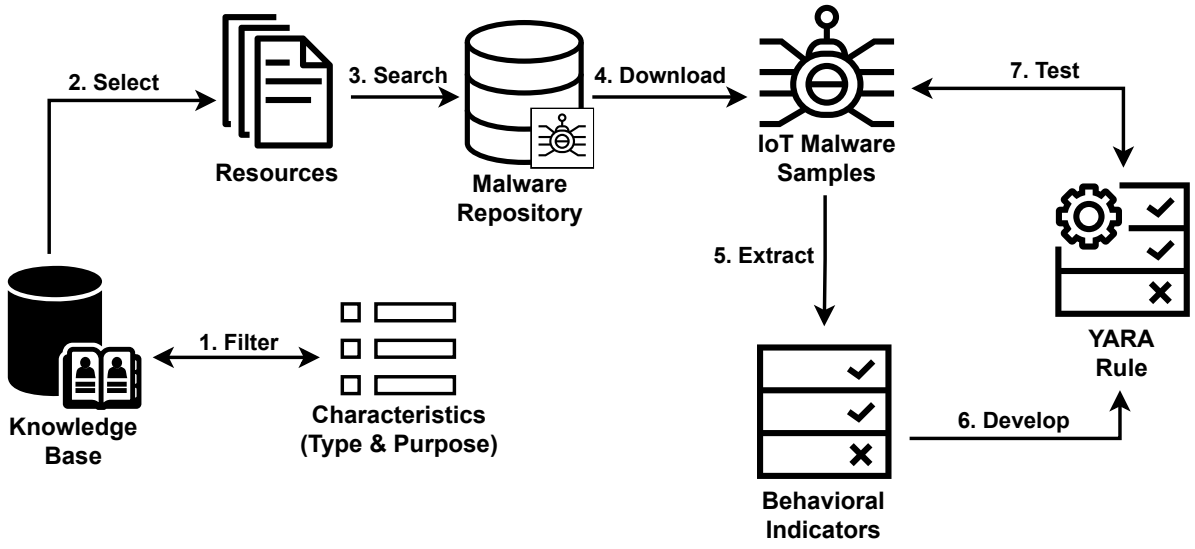

FIGURE 5.2: The diagram illustrates the key steps of the YARA rule development, from selecting the malware variants that match the specific filters to the creation of the actual rule.

5.2 Behavioral Indicator Extraction and YARA Rule Development

This stage focused on developing YARA rules to reliably identify IoT malware based on behavioral indicators. Figure 5.2 provides an overview of the workflow involved in developing YARA rules.

As discussed in the previous chapter, we initially developed a more targeted rule aimed at identifying wiper malware with destructive capabilities. The tools and platforms used during this process were outlined in Section 2.4, but their proper usage is elaborated on in the current section.

To gather the necessary samples for the behavioral indicator extraction process, we used previously gathered resources to extract hashes of known wiper-type malware. Furthermore, we examined the corresponding malware entries on Malpedia, which often contained hashes that linked to online malware repositories. See Section 6.2 for more on the known wiper dataset.

To get the binaries we utilized the VirusTotal CLI's download feature to fetch the files from its repository based on the provided list of hashes. Once we had the necessary samples, we used bincapz static analyzer tool to extract behavioral indicators. This process generated both a human-readable output, with an example shown in Figure 5.3, as well as a more machine-readable JSON format output that could be used for further post-processing. The JSON format output for the same example is included in Appendix B.1. The output provides details about the risk factor of the observed behavior — the higher the risk factor, the more malicious the behavior is, the associated YARA rule that was triggered, a clear description of the behavior, and the specific evidence from the binary sample that matched the rule.

Since we had a limited number of samples, we were able to manually review and select the most relevant behavioral indicators that could effectively distinguish wiper malware from other IoT threats. These indicators, as seen in Figure 5.3, included accessing raw data devices (e.g. generic block device, SD/MMC device), rebooting the machine, and having missing content due to binary obfuscation. The comprehensive details on the behavioral indicators and the corresponding YARA rule will be provided in the following chapter.

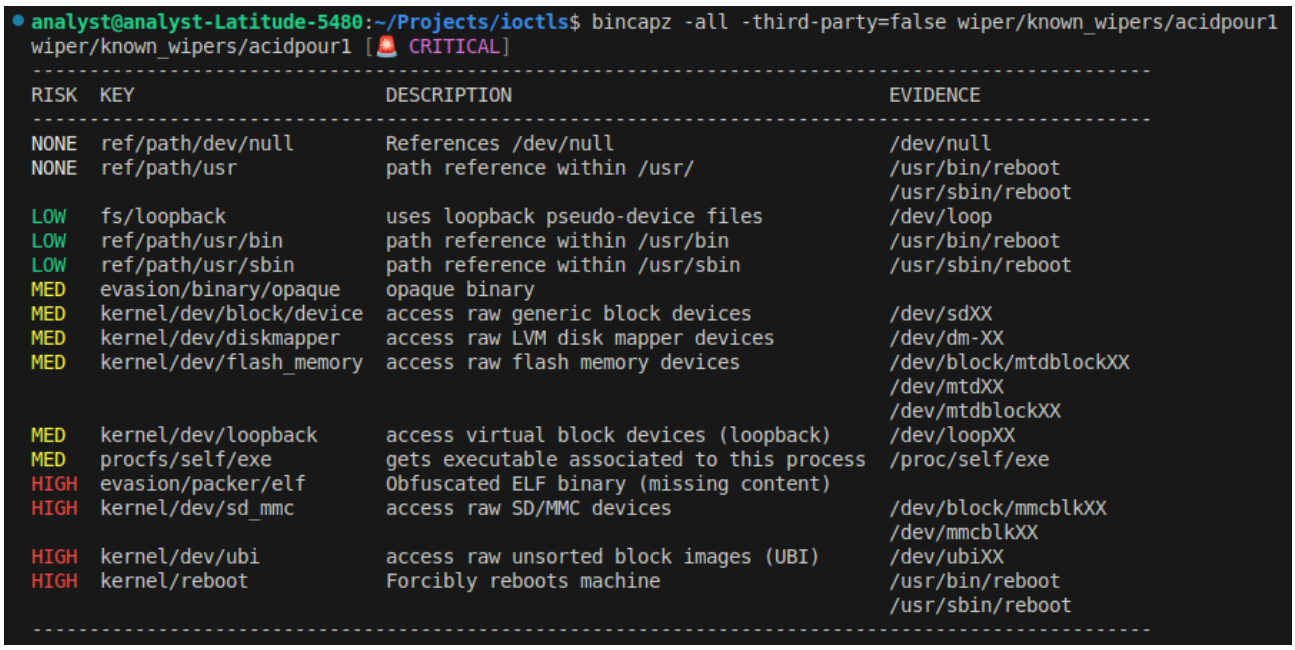

FIGURE 5.3: The figure depicts the output generated by running the bincapz tool on one of the wiper malware samples, specifically the Acidpour variant.

Testing, refining and validating the behavioral indicators was a critical step before proceeding with the development of the YARA rule. We used a combination of techniques to ensure the accuracy and effectiveness of the indicators.

First, we used the corresponding YARA rule from the bincapz repository[10], that would match the selected behavioral indicator. Each rule contained string and/or regex patterns combined in the rule's condition section. To test and refine these patterns we collected strings from the content of the samples, which involved using the strings [11] command-line utility to extract the binary content, and the grep [12] command to match the patterns. Then, we identified additional strings that should be matched, as well as strings that should not be matched based on the specific behavior. The objective was to optimize the rule so that it would only match relevant behavioral indicators, such as function calls or commands related to rebooting, and avoid capturing irrelevant strings like error messages that may contain words like "rebooted" or "rebooting" but are not indicative of the target behavior. This process involved using the regex101[13] web platform as it allowed us to continually test and refine our regex patterns on multiple strings at the same time until we were satisfied with its reliability.

Next, we validated the optimized patterns, we incorporated them one by one into YARA rules and iteratively tested them against the known wiper samples in our list. This method was effective due to the limited number of related samples, which also enabled us to create an Excel spreadsheet documenting the number of matches for each behavioral indicator category across all the samples. The corresponding content of the Excel sheet is shown as a table in Figure 5.4.

[10] https://github.com/chainguard-dev/bincapz/tree/main/rules
[11] https://www.man7.org/linux/man-pages/man1/strings.1.html
[12] https://www.man7.org/linux/man-pages/man1/grep.1.html
[13] https://regex101.com/

Variant	device	access	data	destruct	final	obfuscated
acidpour	7	0	0	0	1	1
acidrain	5	0	0	0	1	1
heh	7	2	2	5	4	0
vpnfilter1	1	1	0	1	1	1
vpnfilter2	1	1	0	1	1	1
vpnfilter3	1	1	0	1	1	1
vpnfilter4	1	1	0	1	1	1
vpnfilter5	1	1	0	1	1	0
handimannypot1	5	0	1	3	3	1
handimannypot2	5	0	1	3	2	0
handimannypot3	5	0	1	3	3	1
handimannypot4	5	0	1	3	3	1
handimannypot5	5	0	1	3	2	0
handimannypot6	5	0	1	3	3	1
handimannypot7	5	0	1	3	3	1
silex1	3	1	1	3	2	0
silex2	4	2	1	3	2	0
silex3	4	2	1	3	2	0
silex4	4	2	1	3	2	0
silex5	4	2	1	3	2	0
silex6	4	2	1	3	2	0
silex7	4	2	1	3	2	0
silex8	4	2	1	3	2	0
silex9	4	2	1	3	2	0

FIGURE 5.4: This figure displays the variant names in the first column, with the corresponding categories of behavioral indicators listed in the row headers. The number of matches for each indicator category is shown in the respective cells. More details about the indicator categories are provided in Section 6.4.

The next step was to combine the refined patterns that captured the relevant behavioral indicators into a comprehensive YARA rule. This rule would then be used to hunt for and identify potential new wiper malware samples.

In case of the more generic YARA rule development, the only differences were in the indicator selection and the validation process, but the overall approach remained the same. The main reason for this was that the number of samples gathered was significantly larger for the more generic IoT malware threat hunting, and the available malware dataset covered a wider variety of malware families. This dataset will be also detailed in Section 6.2.

The indicator selection process involved a combination of manual review and automated techniques. While we still had to manually review the indicators and group common ones into categories as discussed in Section 4.2, we also leveraged Python scripts to extract behaviors across the entire sample set. This approach allowed us to efficiently analyze the indicators across multiple malware families.

The validation process was also adjusted to account for the larger dataset. Instead of an exhaustive per-sample validation, we adopted a sampling-based approach where we selected a subset of samples, typically representing different malware families and variants, to test the YARA rules against.

We also developed a Python script that extracted all strings from each malware sample, identified the associated behavioral indicators and pattern components, and generated an output similar to the bincapz tool. This script provided the flexibility to modify it as needed and use it for additional purposes, such as filtering out samples with few or no matched behavioral indicator patterns. This allowed us to manually review and analyze these samples, as they might reveal new indicators or highlight limitations in our approach. An example of the output of this tool is shown in Appendix C, Figure C.1.

5.3 Retroactive Scanning

After having developed the comprehensive YARA rules for threat hunting, we utilized them to perform a thorough retrospective scan of the VirusTotal malware repository. This scan covered samples that were either uploaded or reanalyzed within a 90-day period prior to the time of the scan.

We were able to upload the rule to the VirusTotal web interface, set the desired timeframe and an email address to receive the completion notification, and then execute the scan against the repository. During the retroactive scanning process, we were able to observe the progress of the scan, including the percentage of completion, as well as the number of matched samples within the specified timeframe. Additionally, the scan provided another count for the matched samples that were outside of this timeframe, effectively covering the entire malware repository. However, this comprehensive reporting feature was only available with a higher-tier subscription plan, which we did not have access to.

After the scan completed, we focused on the matched samples from the specified timeframe. We could access a limited portion of the data through the web interface, but for further analysis, we needed to download the metadata for all the matched samples. Fortunately, the VirusTotal CLI tool offered a feature to fetch all the matched samples by providing the ID of the retrospective hunt scan and specifying the JSON format for the output, which allowed us to efficiently analyze and work with the data.

Having the JSON formated data ingested into our analysis environment, we were able to extract and aggregate various metrics. First we divided them into two lists, one list of detected samples and one list of undetected samples, as we processed them differently. For the undetected samples, we extracted the hashes and metadata in different files, so later we can download the

binary file of the samples and analyze them manually to see whether they are false positives or new threats that need further investigation.

As for the detected samples, we utilized the AVClass tool to automatically categorize them. This allowed us to get a high-level overview of the malware families and variants that were identified by our YARA rules during the retroactive scan. To feed the VirusTotal reports to the tool, we preprocessed the data and created a simplified format that the tool could process. This format included the md5, sha1, and sha256 hashes of the file, as well as the AV labels in a list where each entry is a tuple containing the name of the AV engine as the first parameter and the labels as the second parameter. An example of both the input and output of the tool are presented in Appendix D.

5.4 Real-time Detection

The same pipeline we developed for the Retroactive scanning was also utilized in the most part for the Real-time detection. However, there were some key differences in the implementation as well as in additional purposes that this service could be used for.

Due to the limited number of Retrohunt jobs allowed per month, we developed an alternative approach to test the YARA rules on real-world data, rather than just the collected known malware samples. We leveraged the VirusTotal Livehunt service, which provided us with 25 active rulesets per month that we could also utilize for testing purposes. Whenever we had a YARA rule that was reasonably ready, we would activate it on Livehunt to assess whether it required further adjustment or fine-tuning based on the results obtained. We typically ran these Livehunt tests for a few days to gather sufficient data for analysis.

After gathering the Livehunt data, we iteratively refined the rule by using the Livehunt web interface's testing functionality. This allowed us to simultaneously evaluate the rule against 50 samples by inputting their hashes, and also provided us with an editor featuring autocompletion to ease the use of predefined functions across various YARA modules.

We utilized the testing functionality to both exclude false positive matches from the previous collected data, as well as to identify any false negative cases by testing the rule against samples we knew should be detected. This iterative approach helped us enhance the rule before deploying it again.

Another key difference in the pipeline implementation was how we handled the retrieval of detected sample metadata. In one scenario, the detection notifications were only available for a week, after which they would disappear. This meant that for rules running for longer periods, we had to constantly monitor and save the data to avoid losing it. Additionally, the VirusTotal CLI tool lacked the functionality to save the data in a way that would have suited our needs.

To address this challenge, we developed our own custom script. Despite the API endpoint's limit of 40 samples per request, our script was able to accumulate the metadata for all the samples. It used a cursor-like approach to continue downloading the next batch of 40 samples after each request, until it reached the end of the list or encountered an API resource limitation such as reaching the maximum daily or monthly quota limit. In such cases, the script saved the cursor, allowing us to resume the process when the quota resets. This custom solution provided the same input and output functionality as the method used in the previous section to retrieve the matched samples.

The statistics and results from the Retrohunt and Livehunt analyses are presented in Sections 6.4 and 6.5, covering the wiper malware hunting and the more generic IoT malware hunting, respectively.

Chapter 6

Experiments and Results

In this chapter we discuss the goals and execution of our experiments, the IoT Knowledge Base and other datasets that we gathered during our research, as well as the results obtained from threat hunting and analysis of IoT malware using the approach described in the previous sections.

6.1 Goals

One of the goals of the experiments in this research is to gain a deeper understanding of the characteristics and behaviors of IoT malware variants, which would later be used to enhance static analysis methods, more specifically in developing reliable string and regex patterns to match specific behavioral indicators, thereby reducing false positive matches of the YARA rules.

Another goal is to assess the effectiveness of these YARA rules in a novel threat hunting approach, which involved identifying new unknown IoT malware from the undetected samples, as well as discovering artifacts related to recent cyber attacks or campaigns in the IoT threat landscape based on the detected samples.

One significant challenge we faced during the experiments was the difficulty in manually analyzing the large number of undetected samples uncovered through our threat hunting efforts. While the wiper malware hunting yielded a manageable quantity, the broader IoT malware hunting uncovered a couple thousands of samples, making a manual approach infeasible. As a result, we recognized the need for an infrastructure of automated tools to efficiently analyze this substantial dataset, and we hope that future researchers will build upon this work to address this challenge. We further expanded this idea in Chapter 9.

6.2 Datasets

For the first experiment, we targeted wiper malware by selecting known destructive malware families and variants from our IoT Malware Knowledge Base. These included AcidPour, AcidRain, VPNFilter, Brickerbot, HEH, Silex, and HandyMannyPot. In total, we gathered around 25 samples, comprising one sample each of AcidRain, AcidPour, Brickerbot, and HEH, 5 samples of VPNFilter, 9 samples of Silex, and 7 samples of HandyMannyPot. The list of the gathered known wiper samples is also presented in Appendix E.1.

For the second experiment, we aimed to broaden the scope by including a larger variety of malware characteristics, in order to target more generic set of IoT malware. To achieve this, we collected malware that had been previously deployed by APT groups, as these are typically more sophisticated and feature-rich compared to malware used by cybercriminals. These APT groups include APT41, Volt Typhoon, Black Tech, Zirconium, APT28, Deep Panda, and Sandworm

Team, which was recently designated as APT44 by the Mandiant research group [47]. The dataset consisted of the following number of malware samples and variants: 2 samples of Speculoos, 3 samples of SideWalk, 26 samples of KV-botnet, 3 samples of Hipid, 2 samples of Pakdoor, 12 samples of CyclopsBlink, 10 samples of IoTReaper, 15 samples of Moobot, one sample of ptty, and 160 samples of VPNFilter. This second list of gathered malware samples is provided in Appendix E.2.

To assess the accuracy of our YARA rules, we also needed a dataset of benign samples. For this purpose, we used a large corpus of binaries generated by the Assemblage tool[1], which is a cloud-based system that crawls GitHub for source code, compiles it into binaries, and stores them in a dataset [48]. This dataset comprises approximately 260,000 licensed samples and is publicly available for researchers to conduct further binary analysis.

[1] https://assemblage-dataset.net/

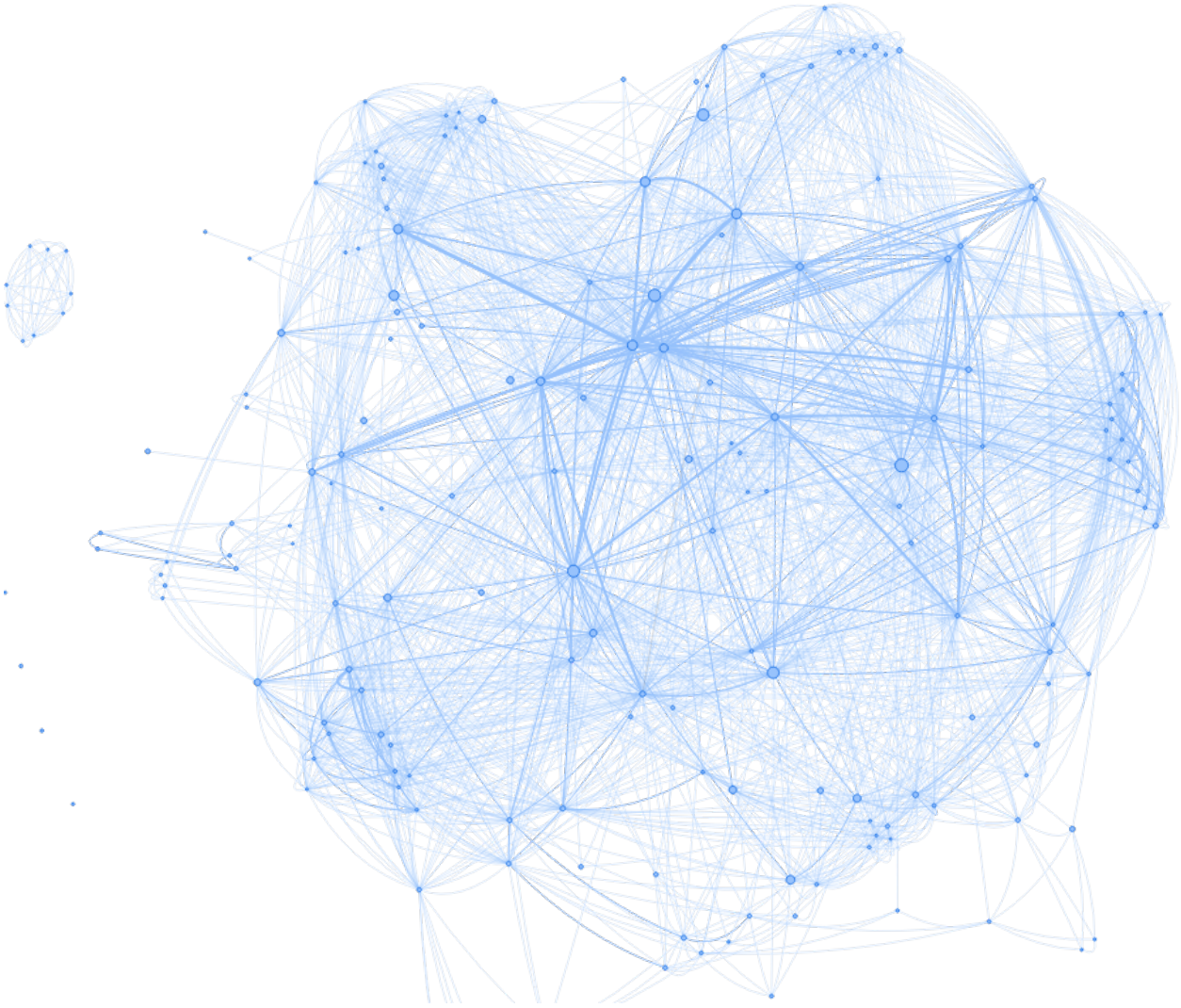

FIGURE 6.1: The figure illustrates the full size of the similarity graph, where the nodes correspond to the malware variants, and their size reflects the number of sources they are associated with, while the edges signify the shared sources between the variants. The small disconnected cluster on the top left side of the graph represents a group of malware that was only present in a single source, while the main cluster in the center of the graph showcases the interconnectedness and overlap between the various domains of resources related to the other malware variants.

FIGURE 6.2: The figure shows a zoomed-in portion of the center of the graph depicted in Figure 6.1. This section includes the following 7 malware variant names: Dark_nexus, Mirai, Moobot, Muhstik, HNS, Persirai, and Satori.

6.3 IoT Malware Knowledge Base

One key component of our research was the creation of an IoT Malware Knowledge Base containing information about previously seen IoT malware families and variants during various past incidents and campaigns. This knowledge base served as a foundation for our experiments, as it allowed us to identify known malware samples and leverage their characteristics for developing reliable YARA rules.

We collected information on a total of 192 IoT malware variants and families, including their characteristics, which were discussed in Section 4.1. This comprehensive process involved examining 513 different sources over a period of approximately 3 months, from the end of January to the end of April 2024.

These sources enabled the creation of relationships between the malware variants. Such relationships illustrate a connection between two variants if they share at least one common domain, where a domain represents the specific host part of the web link of the source. We constructed a graph to represent these relationships, more precisely a similarity graph where the nodes correspond to the malware variants by name, and the edges denote the connections between them. The wider the edge between two variants, the more similar the nodes are which

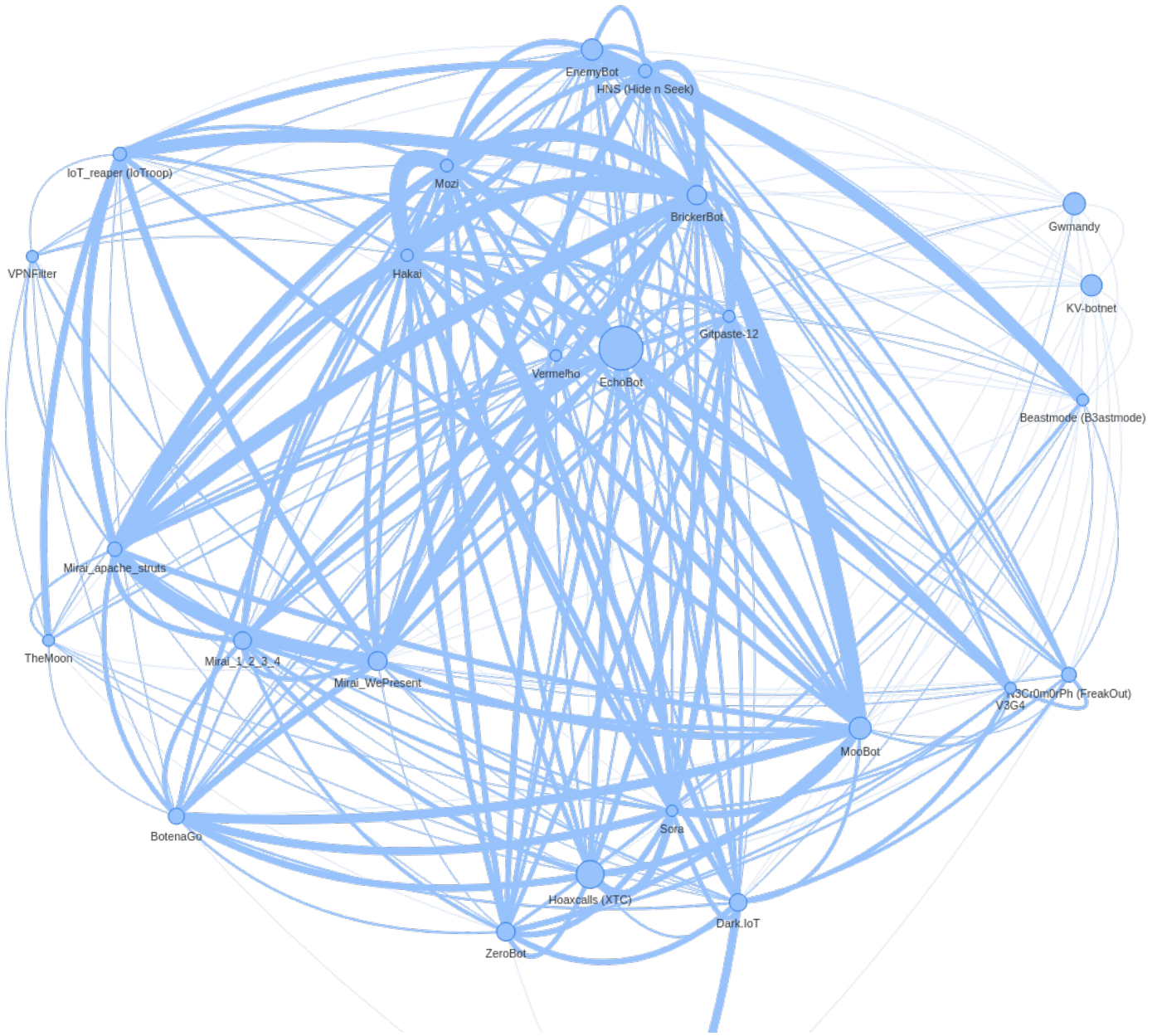

FIGURE 6.3: This figure depicts the similarity graph based on initial access methods, where only malware variants with more than 17 initial access methods are displayed. The value of the threshold was determined to optimize the layout of the nodes and the visibility of their name.

means that they have more sources in common, and the greater the size of the node, the more sources it contains. The complete version of the resulting graph is displayed in Figure 6.1, however, due to its complexity, a more zoomed-in version better illustrates its purpose in Figure 6.2.

The purpose of this representation was to highlight the impact of research institutions, cybersecurity firms, or other entities that publish resources about IoT malware in connecting different campaigns. If an individual reads one of the articles, they are likely to find another article about a different malware variant, as it was published by the same entity on the same domain. This also reflects our research process for the IoT Malware Knowledge Base, as whenever a resource about one variant was encountered, several new variants were discovered that could be further investigated.

Another valuable component of our knowledge base is the detailed list of initial access methods used by each malware family and variant to compromise IoT devices. This information could also be leveraged to represent connections between different malware. We created a graph similar to Figure 6.1, but instead of linking variants based on shared domains, we connected them based on the initial access methods they used. Additionally, we introduced a threshold value to limit the number of displayed nodes and edges, which was determined by the number of initial access methods employed by each variant. The resulting graph is shown in Figure 6.3.

As mentioned in Chapter 4, we encountered a challenge regarding collecting the correct identifiers for each vulnerability used as an initial access method. This process turned into a much larger effort than anticipated and resulted in a list of vulnerable products, vulnerability names, identifiers, associated exploit identifiers, and exploit payload snippets, along with additional notes. In total, we gathered 96 unique vulnerabilities, with most of them successfully linked to their respective exploit identifiers. It is important to highlight that these vulnerabilities were not explicitly mentioned in the resources reviewed, requiring further analysis to identify them. This is a valuable finding that provides useful information for future researchers or security professionals. We have included a simplified version of this table in Appendix F, excluding the exploit payload snippets and additional notes that would not fit within the page

CVE-2023	CVE-2024
CVE-2023-1389	CVE-2024-0769
CVE-2023-25717	CVE-2024-0778
CVE-2023-26801	CVE-2024-3272
CVE-2023-26802	CVE-2024-3273
CVE-2023-27076	CVE-2024-20353
CVE-2023-27997	CVE-2024-20359
CVE-2023-28769	CVE-2024-22768
CVE-2023-32315	CVE-2024-22770
CVE-2023-33617	CVE-2024-22771
CVE-2023-47565	CVE-2024-22772
CVE-2023-48842	CVE-2024-22769
CVE-2023-49897	CVE-2024-23842

FIGURE 6.4: This figure depicts the most recent exploited vulnerabilities with assigned CVE number from 2023 and 2024 that were identified in our research.

CVE	Count
CVE-2017-17215	30
CVE-2014-8361	27
CVE-2016-20016	19
CVE-2018-10561	19
CVE-2016-11059	17
CVE-2016-6277	16
CVE-2018-10562	16
CVE-2015-2051	15
EDB-ID-39596	14
CVE-2018-20062	13

FIGURE 6.5: This figure shows the 10 most commonly exploited vulnerabilities identified in our research.

constraints of this work.

In total, we gathered 557 unique vulnerabilities used by IoT malware, of which 505 were assigned CVE identifiers. The remaining vulnerabilities were cataloged under other platforms such as Exploit Database or Packet Storm Security. Grouping the used CVE identifiers by year reveals that a great number of exploits were associated with 2021, with a total of 122 CVEs. This was followed by 2020 with 81 CVEs and 2017 with 54 CVEs. The most recent vulnerabilities used in IoT malware from 2023 and 2024 can be found in Figure 6.4, while the top 10 list of exploited vulnerabilities over the entire period is presented in Figure 6.5.

Our data collection efforts resulted in additional insights into the characteristics of the IoT malware families. We found that the malware variants could be categorized into different types, with 130 being worms, 44 being backdoors, 20 being droppers, 5 being wipers, and 1 being ransomware. Furthermore, the malware variants displayed various purposes, including 123 with both propagation and DDoS functionalities, 27 with espionage capabilities, 26 with exfiltration features, 20 with cryptomining abilities, 17 that acted as proxy servers, 7 with destructive capabilities, and 2 with white hat objectives. Additionally, when available, we identified the threat actors behind the creation or deployment of the malware. Our research resulted in a list of these actors, which includes well-known APT groups interested in IoT malware, such as APT28, APT31, APT41, and the recently named APT44. Beyond these known APT groups, we also found other threat actor groups and individual cybercriminals involved. In total, we concluded our analysis with 75 unique threat actors. Finally, our investigation showed that many of the identified malware variants were based on or shared common components with previous variants. Specifically, we found that 76 of our variants were derived from the Mirai malware, 16 from Gafgyt, 8 from Tsunami, 3 from Moobot, and 2 from Necro, among others.

To conclude this section, we want to compare our IoT Malware Knowledge Base with the two other studies that surveyed IoT malware. Although Costin's research utilized more resources, 637 compared to our 513, they only identified 60 malware families, while our study uncovered 192 variants. Both knowledge bases share some commonalities, such as the collection of similar information about malware, including vulnerabilities and timeline of events. The study by Princy et al. shared a similar approach to our research, as they also categorized the identified malware based on variant, type, purpose, and other characteristics. However, our knowledge base appears to be more comprehensive in terms of the number of variants discovered, as Princy et al. reported only 77 variants. In contrast, Princy et al. focused more on the targeted devices, including gathering information about the specific hardware and architecture. Additionally, they provided detailed description of all the malware variants and explored machine learning techniques for IoT malware detection, while our knowledge base served as an input for the more important threat hunting component of our research.

6.4 Wiper Hunting Experiment

The first experiment was focused on a more specific behavior - the wiping and destructive capabilities of IoT malware. This allowed us to develop YARA rules to hunt for active wiper malware targeting IoT devices.

After analyzing the 25 wiper malware samples, identifying and optimizing the behavioral indicators, we developed and iteratively refined the wiper YARA rule with various strings and regular expression patterns to match different behavioral indicators. We grouped the indicators into several categories, including:

- Hard-coded paths to raw generic block devices, raw SD/MMC devices, raw LVM disk mapper devices, raw flash memory devices, raw unsorted block images, and raw system memory that could be targeted for malicious overwriting.

- Hard-coded paths to files like `/dev/zero` and `/dev/urandom` that could be used to overwrite data.
- Commands such as `dd`, `cp`, `cat`, `fdisk`, `truncate`, and `rm` that can used for malicious file operations like overwriting, reformatting, truncating, or deleting.
- The special destructive `rm -rf` command that can forcibly remove files and directories.
- Final commands that could lead to devices halting, restarting, or shutting down.
- Lack of essential strings that could indicate code obfuscation.

The YARA rule was refined through 5 iterative rounds. The final, fifth version of the rule is provided in Appendix G.1. Each version of the rule was validated by running it on either Retrohunt or Livehunt services.

The first version of the YARA rule was only built based on our analysis of an initial set of 10 wiper samples. This rule matched 10,000 samples on a Retrohunt job. The results included various file types, such as Windows, Linux executables, and others. To improve the rule's precision, we realized the need to apply file type filtering. Before moving to the next iteration, we locally filtered the matched samples and found 1,597 ELF executables. Of these, approximately 94% had at least one detection on VirusTotal, and around 93% had at least 25 detections. Despite the 3-month time frame for the Retrohunt job, we noted that the full list of matched samples from this period may not have been captured due to the lack of applied filtering and the 10,000-sample limit. However, we identified 95 files with no detections and an additional 12 with a low number of detections.

The second version of the YARA rule incorporated the file type filtering, and it still reached 10,000 samples. However, this time, all the matched samples were the correct file type. The results showed that 8,375 samples had at least one detection on VirusTotal, while 1,625 samples had no detections. This version provided better results in terms of identifying IoT malware, as verified by examining the metadata from VirusTotal, including the `popular_threat_classification.suggested_threat_label` field, which gave insights into how AV solutions had categorized these samples. Although the majority of the detected samples were identified as belonging to common IoT malware families like XORDDoS, Mirai, and Gafgyt, we had not yet found the specific wiper malware samples we were looking for.

Seeing promising results from the second version, in the third iteration we refined the YARA rule to be stricter and focus more on wiper malware behaviors. We re-examined the original wiper samples and retested the effectiveness of the stricter patterns. We also revised the behavioral indicators, removing those that were difficult to reliably detect due to evasion techniques used by malware, and adding new ones that appeared more effective for reducing false positive matches. Additionally, we decided to search for other file types beyond just executables, such as scripts and C programs, and applied a file size limitation. We tested this updated rule on the Livehunt service for about 10 days from 3rd of May till 13th of May 2024. The rule matched 115 samples, of which 63 had fewer than 23 AV detections and 52 had 23 or more detections. We selected 23 as the detection threshold based on observations from a histogram analysis of the matched samples' detection rates. Further investigation of the high-detection samples led to the identification of an additional Silex wiper sample not included in the initial 10 samples used to create the rule. This discovery prompted a deeper look into the new sample, resulting in the identification of 7 more Silex wiper samples, expanding our known wiper dataset to 18 samples. Additionally, a review of the IoT Malware Knowledge Base revealed a previously untagged malware family, HandyMannnyPot, associated with destructive purpose. Examining the collected data for this family uncovered 7 more samples, increasing the total number of known wiper samples from 18 to 25. While analyzing the low-detection samples,

we encountered a bash script that was used, to our understanding, to harden VitalChat devices. This demonstrated that although the match was a false positive, the observed behavior could potentially be malicious if it was not implemented by the device manufacturer.

Building upon insights from the previous iteration, we created the fourth version of the YARA rule by refining it to account for the newly discovered wiper samples. In addition to that, we utilized the low-detection samples from the prior round to further enhance the rule through Livehunt testing functionality. During this process, we established the four conditions also present in the final rule. Using this version of the rule, we run a 3-month Retrohunt job from February 16th to May 15th 2024, resulting in 192 matched samples.

From our quick analysis of the samples, we identified two previously unknown wiper variants. Five of the samples contained the distinct string `KADENBOTNET`, indicating the presence of a new KADEN botnet. These five samples are the first five entries on the list shown in Figure 6.6. The sixth sample on the list was identified by searching for the specific string on VirusTotal's Intelligence Search feature. This search also resulted in additional samples containing the same string, but these did not have the wiping capability and had been uploaded to the platform prior to our investigation. This list of additional samples is provided in Figure 6.7.

Furthermore, the analysis of the 192 samples revealed the presence of several strings associated with the LOLFME malware [49, 50], which was created by the KekSec cybercriminal group. These strings were found in three additional samples suggesting the existence of a new variant of the LOLFME malware with wiping capabilities. The hashes related to these samples are listed in Figure 6.8.

All the eight samples were first uploaded to VirusTotal at the same time as our investigation was ongoing. We could not find any prior references to the `KADENBOTNET` string or the KekSec malware with wiping capabilities, suggesting that these were new discoveries. Additional details about the newly identified wiper variants were published in a blog post [51] based on findings from this research.

Even though we found new wiper variants during the Retrohunt process, a significant portion of the 192 matched samples appeared to be false positives upon further investigation. Furthermore, we encountered another challenge, as the rule failed to match the VPNFilter variants from our known wiper dataset. To address this issue, we created a fifth and final version of the YARA rule that included a fix. This fix involved modifying one of the conditions to detect the use of the special, destructive `rm -rf` command, which allowed it to successfully match all 25 known wiper samples. Additionally, the final rule eliminated any false positives from the previous versions. This paved the way for the concluding Livehunt run from May 17th to June 27th 2024, which resulted in 32 matched samples. The matches included previously identified malware from the known wiper dataset or prior iterations of the investigation, as well as newly discovered samples of Silex, KADEN, and LOLFME malware. These new samples are listed in Figure 6.9, along with their corresponding file hashes.

To conclude this experiment, we assessed the accuracy of the final wiper YARA rule by running it against the Assemblage dataset discussed in Section 6.2. The outcome was a 100% accuracy rate - none of the binaries in the dataset were incorrectly matched, and the rule successfully identified all the known wiper samples in our own dataset.

Variant	sha256
KADEN	425763146eb6a8a5f2da5d481b1752806032e53d83b304d08c15b010c3578508
KADEN	b90a6b309ee29d3633a5ed9b49de78e38ce68f44a4d014977b3f8322c76a4d2d
KADEN	44e7fef792b63debd87e8eb9636675deea29febe0107a9f708adbe5bdca1dead
KADEN	cb139f5abd6ce16191b40ee01b0f1b10c9846469b265fab0a972bd0388a838f8
KADEN	6103262a245fb296edf2f5c0ffc99e5a74bc4b267d304a92c934e37045811d93
KADEN	0cb0872edf98d32320328a92b2d1a563ecdcea398866d0b3f2b67b016b010636

FIGURE 6.6: This figure showcases the five wiper samples identified as part of the KADEN botnet during the Retrohunt job, as well as one additional sample with wiping capabilities that was discovered through a manual search using VirusTotal's Intelligence feature.

Variant	sha256
KADEN	1939dce51318d20c9b48b71421882e31bc11a199f0001bd33a27086f0a17d909
KADEN	c557a99d78a0adbcf5e040f71458d682d3093fd8ac3e2624b7a780a9a9e5d1bd
KADEN	bf3b5884b6204194a983bb088ba58ba5ef9b88cdaa7c852640bd67e00619d5ed
KADEN	a6a35cc9336cd6db225ef3617855fec1117b78c778c5655192c125107e4f58e2
KADEN	86ed7f04a518926ce776376a374fcd9245e638687b9db11e8c84abe484c7159c
KADEN	9fa8c4e982e6f094ba481758b2065ce1ccd5e58aa39809059af9b558eca39cc3
KADEN	ce1ea5ab42c12484a73ea403eba2c69a316f1e03371f9d313ac64fb61f1f8cff
KADEN	11711729a95e4f4d9627d4bfdc955d3cd26e4c75751f3a90a96a0f89623ccdab
KADEN	e0dd29ada021b45d8e5e17d8851f2d7fd320685aa8bbbbb124428b8918f6ac30
KADEN	fb01be79fcb5489c5889c2910b55ee697bb1a544735308ab4ca5c2a26e7925ed
KADEN	5a19f153bbd0afdf3abf83340a4c4c467fffeda09a470d7d62176265ab7bcff3
KADEN	89211225d0afe0c86fa4dc0017559e3025c3d195dbf5f53b684d9f39bcab1867
KADEN	68f25f6b1fb02052f9e53ef86404ab733034633a0682ec13e5b80c7c80043e21
KADEN	49addddb7fbfa9717a0f4cabe88dc0570778eee33a086b2e2c8bfdf9d91eec68

FIGURE 6.7: This figure contains additional samples that were found to be related to the KADEN botnet but did not have the wiping capability.

Variant	sha256
LOLFME	ff33ce4ea04cf26ad62ca72e6b3072485ed6a5e16ee981cb471bd649b12f9494
LOLFME	26a494382bbfa16e8674beee16c89e5704b8abd1e1283b0fa28ec2d9d7bfebd9
LOLFME	0b471ee1ad869b54b12efcd13ef2a024c555232a814b021c119de5e7a63789bf

FIGURE 6.8: This figure lists the three new LOLFME malware variants with wiper capabilities that we discovered during the Retrohunt process.

Variant	sha256
LOLFME	50bdb7eee2d3f36346366fc3f2b6f6e66a24268f78e46b678fec746198715845
KADEN	e214ae9fc129d5bb3e5d9f08435d98cc8d41a5e4d84b5e95353fb6faa1720825
KADEN	735db56fc9b889422c0cc2921438812fb4b0f54e17c47ca03327880ba587f8e1
Silex	7be62c81b9a4165ceb456e9f83a82d5c4fe9a66a149592172290f6ac41c954da
Silex	9d3962c41f179e012d09178ee356bdd4b23ad1681911c571e363ebaed706cf2b

FIGURE 6.9: This figure presents a list of newly uncovered wiper samples that were identified by running the fifth version of the wiper YARA rule on the Livehunt platform.

6.5 Expanded Hunting Experiment

The second experiment aimed to move beyond the focus on wiper capabilities of malware and explore a more generic approach to IoT threat hunting. To achieve this, we developed a YARA rule that targeted multiple sets of behavioral indicators of IoT malware, rather than a specific one.

We analyzed the 234 IoT malware samples deployed by APTs, which were introduced as the second dataset in Section 6.2. From the analysis, we identified common behaviors among the different variants, which allowed us to develop rule conditions targeting these shared characteristics. These shared characteristics included the following:

- Indicators related to command-and-control communication, such as the use of C socket functions, BIO or SSL/TLS socket functions, and references to proxy client/server and tunneling.
- Network-related behaviors, including lookups, formatting, parsing, and iptables command usage.
- Cryptographic indicators, such as the presence of certificates, references, cipher usage, and various public, private, and secret keys.
- Evasion techniques, including obfuscation, masquerading[2], and hiding on the system.
- Execution of shell scripts and programs.
- Persistence mechanisms, such as modifying startup files, crontabs, or SSH configurations.
- File dropping operations, which may involve URL construction, HTTP requests, and different fetching commands.
- Spreading tactics, like using nmap, enumerating networking interfaces, and accessing files, such as `/var/log/lastlog`, that could contain sensitive login information.

Using the identified behavioral characteristics, we developed a modular YARA rule capable of detecting a wide range of IoT malware families. This rule is attached to Appendix G.2, and it consists of eight conditions, each targeting specific variants such as CyclopsBlink, Hipid, Speculoos, IoTReaper, KVBotnet, Moobot, SideWalk, and VPNFilter.

It is important to note that the VPNFilter samples with wiping capabilities were excluded from the behavioral extraction phase as they lacked sufficient number of behavioral indicators.

[2] https://attack.mitre.org/techniques/T1036/

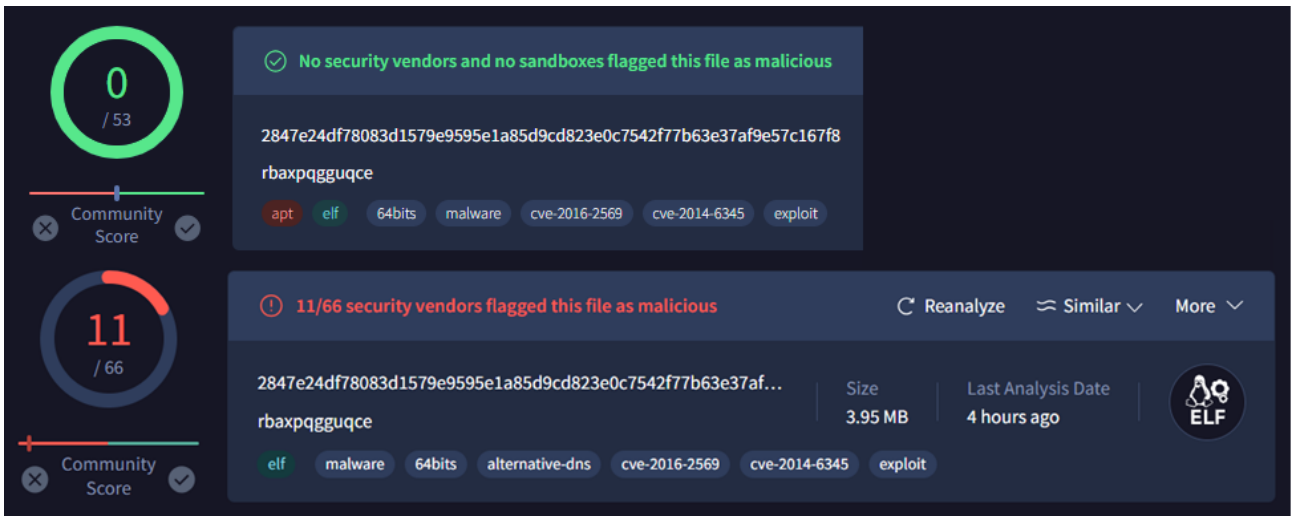

FIGURE 6.10: The figure shows the VirusTotal detection rate history for the interesting sample we discovered during the initial Livehunt run of the more generic YARA rule.

Additionally, we filtered out samples that did not meet the same criteria leaving us with the previously mentioned 234 samples instead of the initial 277 total.

This time to create the final version of the rule, we only used two iterations, as we had incorporated the learning from the previous hunting part of the research. The initial version was meant to test whether the concept of grouping the behavioral indicators would work effectively and match primarily IoT malware. This first version of the YARA rule included only a portion of the conditions, targeting five of the previously mentioned eight variants. Additionally, the behavioral indicators were limited in number, excluding those related to cryptography, persistence, spreading techniques, and file dropping.

We left the first version of the rule on Livehunt running for 19 hours, matching 1869 samples, out of which 412 had 0 AV detections, while 1457 had atleast one detection. Unfortunately, we made a mistake not including filtering on file size and type, which we had to locally do. We set the file type to ELF executable and the size threshold value to 5.3248MB. The optimal threshold value was selected based on multiple values being tested and finding the most effective one to remove undetected samples, without removing too many detected ones. After the filtering we were left with 1290 detected samples and 95 undetected ones.

Upon examining the 95 undetected samples, we identified one particularly interesting sample with the SHA256 hash `2847e24df78083d1579e9595e1a85d9cd823e0c7542f77b63e37af9e57c167f8`. This sample had no AV detections on VirusTotal when we first encountered it on June 18th 2024, but by July 24th 2024, it had acquired 11 detections, categorizing it as a Trojan. The detection rates are shown in Figure 6.10, showcasing also the fact that our initial scan tagged the sample as "apt". This indicates that our YARA rule had matched it while it was still active. A closer static analysis of the sample revealed a list of 32 hard-coded IP addresses, which could suggest either a list of potential targets or a list of command-and-control servers or relay IPs. However, further investigation using VirusTotal's search functionality showed that none of these IP addresses had any malicious activity tied to them. Additionally, it included 26 strings indicative of fuzzing capabilities targeting a PHP web server, as well as a comprehensive list of 41 different user agents, likely used to enhance its stealthiness by mimicking a variety of user agents. Furthermore, the sample contained a public key and behavioral indicators related to the use of a proxy server.

During the Livehunt run, we closely monitored the detected samples to identify any interesting malware that could be associated with IoT devices. We analyzed samples by chance, reviewing their VirusTotal reports and conducting static analysis on them. In the initial matches,

we discovered some intriguing samples that were classified as Rekoobe malware [52] - a Trojan Linux bot capable of infecting machines with diverse CPU architectures, including SPARC, Intel x86, and x86-64. The hashes of these two samples were `cac63e105d73d59c7f83779005ada0a4d3f7fb072cfc2c9590b64fe3896d2e3e` and `5837c2340b701256f2988dffa8bf19211f92d97bbdd4c79ea9ef15b9c8c1a822`. Interestingly, these samples were not mentioned in any of the articles or platforms, such as Malpedia[3], that we had previously reviewed, suggesting that our findings could provide valuable insights to the security community.

The Rekoobe samples we discovered appears to be part of a campaign conducted by the Chinese nation state-sponsored threat group APT31 [53], which has been targeting South Korea for several years. Basing our initial behavioral indicator extraction on APT-deployed malware turned out to be a wise choice, as it enabled us to discover these new samples, offering valuable insights into IoT malware related to sophisticated threat groups.

In addition to the Rekoobe samples, we also discovered a Mirai variant called Faithful or Fall, named after the file naming convention used by the threat actors. The sample with the hash `9b34020669f9c18d3f4b2d0f67dffceacd9ef668b5656e0140279df6d268f4a7` exhibited behavioral indicators for file dropping operations, including fetching the appropriate executable based on the target device's CPU architecture. Moreover, the sample contained spreading capabilities that leveraged the Huawei RCE vulnerability from 2017 [54] to infect other machines.

After analyzing additional samples, we decided to refine the YARA rule further by incorporating the remaining behavioral indicator categories and conditions. The grouping of the behaviors proved to be promising and effective in detecting a broader range of IoT malware, as demonstrated in the prior analysis. Consequently, we revised the rule to include the complete set of behavioral indicators, targeting all eight variants, and prepared to test the final rule over a longer duration.

The refined YARA rule was deployed on Retrohunt across three consecutive runs, each covering a different and shorter timeframe between March 28th and June 25th 2024, and once on Livehunt from June 26th to July 2nd 2024, for a total coverage of 97 days. This resulted in 29,699 samples matched on VirusTotal, of which 27,469 were detected by AV engines and 2,230 were undetected. We recognized that manually analyzing such a large number of undetected samples was not feasible, so we decided to focus on the detected samples, reviewing them through an automated approach. Moreover, we discuss the limitations regarding the undetected samples in Chapter 8, while we also propose a related theoretical solution in Chapter 9.

For the detected samples, we automated the process of further analyzing them by first extracting the VirusTotal reports and preprocessing them into a specific `lb` format that the AVClass tool could recognize. This allowed us to obtain the malware family names and the corresponding malware classes identified by the AV engines. The preliminary analysis of the AVClass output revealed a diverse range of IoT malware families and the corresponding malware classes they belong to, with the number of occurrences for each entry shown in Figure 6.11 and Figure 6.12.

To further analyze the identified malware families, we examined whether any of the findings matched known IoT malware campaigns conducted around the time of the detections. By cross-referencing the malware families with information from IoT-related publications, blogs, and platforms like Malpedia, we found potential connections. Specifically, the Tsunami[4] malware was also referred to as Muhstik in a blog post targeting message queuing services [55]. Additionally, the sshdoor and sshdkit entries were similar to the Ebury[5] malware, which is known to compromise systems through SSH [56], suggesting a possible connection to these identified families as well.

[3] https://malpedia.caad.fkie.fraunhofer.de/details/elf.rekoobe
[4] https://malpedia.caad.fkie.fraunhofer.de/details/elf.tsunami
[5] https://malpedia.caad.fkie.fraunhofer.de/details/elf.ebury

These findings could offer valuable insights into the threat landscape of IoT malware and the campaigns targeting these devices. Furthermore, our research approach could be leveraged to gather timely, up-to-date, and actionable threat intelligence that can enhance security measures.

In conclusion, we evaluated the performance of the refined, more generic YARA rule using the same dataset employed for testing the wiper-specific YARA rule. Our findings indicate that this rule also demonstrates a high degree of accuracy, matching only 2,138 benign binary files out of the total 259,016 files present in the dataset, resulting in an approximate 99% accuracy rate.

Family Name	Count
mirai	14664
gafgyt	11938
xorddos	9826
kaiji	693
chaos	623
lightaidra	473
tsunami	284
sshdoor	111
sshdkit	74
mozi	70
setag	68
ddostf	62
mrblack	60
dofloo	44
rekoobe	43
beebone	29
winnti	29
xmrminer	24
elknot	17
chinaz	15

FIGURE 6.11: This table showcases the top 20 most prevalent malware families identified by the AV engines on our matched dataset of samples, with Mirai being the most dominant, followed by Gafgyt, XorDDoS, and Kaiji.

Class Name	Count
backdoor	15227
grayware	444
miner	301
grayware:tool	263
downloader	242
miner:bitcoinminer	179
grayware:adware	65
ransomware	40
virus	17
grayware:tool:nettool	8
worm	8
bot	7
rootkit	7
grayware:tool:remoteadmin	5

FIGURE 6.12: This table illustrates the malware classes identified by the AV engines on our matched dataset of samples, with the most prevalent being the backdoors.

Chapter 7

Related Works

This chapter provides a summary of the previous studies and methodologies that are significantly related to malware detection and analysis. Various methods and techniques have been proposed for detecting and analyzing malware. These can be broadly categorized into traditional and learning-based methods [16].

7.1 Traditional Methods

Traditional methods, such as signature-based detection methods, identify known malware samples by analyzing static features like specific code patterns or signatures. Some common static analysis techniques for detecting malware include: binary analysis, opcodes analysis, permission-based detections, ELF-Miner, and graph-based detections [33].

While signature-based detection method is effective for detecting known threats, it struggles with identifying new or unknown malware variants. For instance, Ngo et al. highlight the limitations of signature-based methods against obfuscation techniques employed by modern malware [33]. An alternative approach is to use behavioral-based detection method, another traditional one. Behavioral analysis focuses on observing and analyzing the runtime behavior of a program to detect malicious activities. This contrasts with static analysis, which examines the code without executing it. Some common dynamic analysis techniques used in behavior-based malware detection include: sandboxing, full system emulation, process monitoring, and dynamic data flow analysis [17].

Dynamic analysis techniques can be used independently or in combination to provide a comprehensive view of a program's behavior. For instance, a security solution might use sandboxing to initially detonate a suspicious file and then employ process monitoring to log its activities for further analysis.

7.2 Learning-based Methods

Unlike traditional methods that rely on predefined rules or signatures, learning-based malware detection techniques use machine learning and deep learning models to automatically learn patterns and features from data to identify malicious patterns [16]. This allows them to adapt to new and evolving threats more effectively.

Several common machine learning algorithms have been used for malware detection, including: K-Nearest Neighbors, Naive Bayes, Random Forest, and Support Vector Machines.

Machine learning models can be trained and used on static features extracted from malware samples, such as the opcodes or the visual representations created by converting the malware binaries into images. Then they can detect potentially malicious behavior by identifying unusual

patterns in the sequences or combinations of opcodes. Additionally, they can compare the visual representations of analyzed samples to the training data, allowing them to detect similarities with known malware families, as malware samples from the same families often exhibit visual similarities in their underlying structure.

On the other hand, deep learning models like Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) can also be effectively utilized for malware detection. These models automatically learn relevant features from raw data, reducing the need for manual feature engineering. RNN models, in particular, are effective for analyzing sequential data like API call sequences or network traffic. They can learn temporal dependencies and identify anomalies. Additionally, CNN models, commonly used for image recognition tasks, can be applied to malware detection by analyzing binaries as images.

Chapter 8

Limitations

In this chapter, we discuss several limitations encountered in this research. Our primary limitations stemmed from the challenges of analyzing undetected malware samples and the constraints of utilizing online malware repositories.

One key limitation was our inability to manually examine the significantly increased volume of undetected samples in the second experiment, exceeding what could reasonably be examined manually. While our YARA rules demonstrated effectiveness in identifying IoT malware, we believe they miss samples that employ evasive techniques. These samples could be packed, obfuscated, or have essential components stripped, which can prevent the YARA rules from successfully matching the behavioral indicators. Consequently, such samples may evade detection by our current approach. These evasive methods hinder the extraction of meaningful behavioral indicators, thereby impairing our ability to detect such samples with YARA rules and resulting in failure to identify those samples. This highlights the inherent limitation of signature-based detection approaches.

Beyond the challenges inherent to malware, we also faced limitations with the YARA rule engines and our selected online malware repository, VirusTotal. For instance, including hard-coded IP address matching in our YARA rules proved impractical due to performance issues within the VirusTotal YARA engine, which restricted our capacity to identify malicious samples based on such network-based indicators. Additionally, the VirusTotal services also imposed several constraints:

- Limited Intelligence Searches: With only 300 intelligence searches allowed per month, both individual hash lookups and broader intelligence queries were restricted, impacting our ability to out of the box gather comprehensive threat intelligence. To address this limitation, we utilized the VirusTotal API tool as the only viable solution.

- Retrohunt Limitations: Retrohunt jobs, which allow retrospective analysis of malware, were limited to a 90-day timeframe. With only two Retrohunt jobs available per month, our ability to investigate past campaigns was significantly restricted, however, we attempted to divide our analysis efforts across multiple months and relied more on Livehunt ruleset when possible.

- Livehunt Limitations: Livehunt, which provides real-time detection notifications, only retained these notifications for one week. This necessitated constant monitoring and introduced the risk of missing crucial detections.

- API Lookup Quotas: Accessing VirusTotal data through their API was restricted by quotas like daily 1000 lookups and monthly 30000 lookups. This limitation significantly constrained large-scale analysis as they limited our ability to download matched samples in bulk. To work around this constraint, we had to come up with a script that downloads

the samples with a cursor-like method. If a quota limit was reached, it would save the cursors position and would allow us to resume from where the script left off after the limit window expires.

These limitations highlight the challenges we faced while dealing with large datasets and the reliance on third-party platforms. Future research could explore alternative malware analysis platforms or develop more sophisticated techniques to circumvent the limitations of YARA rules and enhance the detection of highly evasive malware. This will be further discussed in the next section.

Chapter 9

Future Works

In this chapter, we outline potential future research directions that could enhance the effectiveness of our approach to IoT malware detection and analysis. These directions aim to overcome the limitations discussed in the previous chapter, as well as identify new areas of exploration for the research community.

Automating and scaling up the processes of malware collection and analysis could be a promising approach to explore further. Leveraging techniques like an automated pipeline incorporating malware metadata and sample retriever, behavioral indicator extraction and machine learning-based behavioral classification could enable more comprehensive and efficient analysis of larger volumes of samples, addressing the challenges posed by the significantly increased number of undetected samples. Moreover, malware samples employing evasive techniques to hinder static analysis could potentially be detected by an automated multi-step process. This could involve chaining the use of advanced unpacking and deobfuscation tools, such as the FLOSS[1] tool developed by researchers from Mandiant [57], to automatically extract key artifacts from the obfuscated malware samples and enable further static analysis.

Another potential approach is to further explore dynamic analysis techniques, as they have the potential to uncover behavioral indicators that static analysis may miss, and reduce the number of false positive detections ensuring that the undetected samples identified by the rules are confirmed to be malicious. We propose implementing an extension to the Drakvuf tool mentioned in Section 2.3. This extension should support analysis of diverse CPU architectures beyond x86 and ARM. Given the wide range of CPU architectures found in IoT devices, a solution capable of handling multiple architectures would be highly beneficial. For instance, the MIPS CPU architecture used in the VPNFilter [58] malware is an example of an alternative architecture that could be addressed through such an extension.

While automatic solutions would greatly enhance malware analysis, a challenge still persists. These solutions require the malware sample to be first identified and downloaded before analysis can occur. This raises the question of how to efficiently detect the presence of malicious samples that use evasive techniques in the first place. One potential approach would be to further explore the use of VirusTotal metadata, specifically identifying common metadata patterns associated with known IoT malware. This approach could enable the detection of IoT malware samples without necessitating the download and analysis of the binaries. However, these analysis methods would still be employed to validate the metadata-based approach and develop the necessary detection rules.

As the use of YARA rules is becoming deprecated due to the introduction of a more advanced version called YARA-X [2] [59], further research into YARA-X could present an opportunity to improve the rule-based detection mechanisms. The new version could potentially overcome the

[1] https://github.com/mandiant/flare-floss

[2] https://virustotal.github.io/yara-x/docs/intro/yara-x-vs-yara/

limitations observed in this study such as the performance challenges regarding the current YARA rule engine within VirusTotal.

Lastly, an interesting research direction could be to explore the application of YARA rules coupled with behavioral indicators in domains beyond just IoT malware, such as in the OT or healthcare sectors. These sectors often involve additional adversary tactics and techniques that differ from the typical deployment of malicious binaries, such as receiving documents containing embedded macro malware or phishing emails with malicious links. This research could entail examining the prevalent attack vectors, the file types utilized, the behaviors exhibited by these malicious files, and the resulting indicators that can be observed through further analysis. Applying our research approach to other domains could provide valuable insights for the broader security community.

Chapter 10

Conclusion

This research paper has outlined a comprehensive approach to threat hunting and analysis of IoT malware. We described the goals and methodology of the study, followed by a detailed overview of the processes and tools utilized, and presented the experiments conducted along with their results. Additionally, we highlighted the challenges and limitations encountered, and discussed potential future research directions to address these issues.

The primary goals of this study were to build a more comprehensive knowledge base of current IoT malware, as well as examine IoT malware samples to identify unique signatures or characteristics that could be leveraged for reliable IoT malware detection.

To achieve these goals, we have created an IoT Malware Knowledge Base comprising 192 entries with detailed information on malware variants and their characteristics. In addition, we have crafted YARA rules based on behavioral indicators, enabling the detection of both new samples of known IoT malware and their emerging variants. During the development of detection rules, we presented a novel approach to identify malware on online repositories such as VirusTotal, leveraging the previously mentioned indicators as well as metadata provided by the platform. This process also necessitates systematization to facilitate larger-scale utilization. We have also validated and evaluated the effectiveness of the developed YARA rules by testing them against an extensive dataset of benign binaries and another dataset of pre-assembled collection of known IoT malware samples.

Our approach has shown promising results, including the identification and categorization of a diverse set of behavioral indicators through the conducted experiments. Additionally, we have constructed both a smaller, specialized dataset focused on wiper malware, as well as a more comprehensive, generic IoT malware dataset comprising malicious samples. Besides these datasets of known IoT malware, we discovered the KADEN botnet and a new wiper variant of the LOLFME botnet in the first experiment. While in the second experiment, our YARA rules matched a wide range of IoT malware families, later classified by AVClass tool, and it also matched a significant number of promising samples that were undetected by AVs, requiring additional analysis in a future research. Although we consider this a success, we have identified several limitations and challenges.

In conclusion, this study has made significant contributions to the field of IoT malware threat hunting and analysis, providing a comprehensive knowledge base, novel detection mechanisms and valuable datasets, as well as encouraging researchers to explore potential improvements to our method and future directions.

Bibliography

[1] Jie Ding, Mahyar Nemati, Chathurika Ranaweera, and Jinho Choi. Iot connectivity technologies and applications: A survey. *IEEE Access*, 8:67646–67673, 2020. doi: 10.1109/ACCESS.2020.2985932.

[2] Syed Kashan Ali Shah and Waqas Mahmood. Smart home automation using iot and its low cost implementation. *Int. J. Eng. Manuf*, 10(5):28–36, 2020.

[3] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, Shanay Rab, and Rajiv Suman. Upgrading the manufacturing sector via applications of industrial internet of things (iiot). *Sensors International*, 2:100129, 2021. ISSN 2666-3511. doi: https://doi.org/10.1016/j.sintl.2021.100129. URL https://www.sciencedirect.com/science/article/pii/S2666351121000504.

[4] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22 – 32, 2014. doi: 10.1109/JIOT.2014.2306328. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84908397157&doi=10.1109%2fJIOT.2014.2306328&partnerID=40&md5=a26ce02f24ad18f54727d982b99fd161. Cited by: 4437; All Open Access, Bronze Open Access, Green Open Access.

[5] Bill Bither. What is an edge device and why is it essential for iot? https://www.machinemetrics.com/blog/edge-devices, January 07, 2021. Accessed: April 8, 2024.

[6] CUJO AI. 15 most popular iot products and devices in 2021. https://cujo.com/blog/15-most-popular-iot-products-and-devices-in-2021/, July 27, 2021. Accessed: April 8, 2024.

[7] Palo Alto Networks. What is the difference between iot and ot security? https://www.paloaltonetworks.com/cyberpedia/iot-security-vs-ot-security, 2024. Accessed: April 8, 2024.

[8] Viral Gandhi. 2023 threatlabz report indicates 400 https://www.zscaler.com/blogs/security-research/2023-threatlabz-report-indicates-400-growth-iot-malware-attacks, October 24, 2023. Accessed: April 8, 2024.

[9] Securelist by Kaspersky. Overview of iot threats in 2023. https://securelist.com/iot-threat-report-2023/110644/, September 21, 2023. Accessed: April 8, 2024.

[10] Elie Bursztein. Inside mirai the infamous iot botnet: A retrospective analysis. https://elie.net/blog/security/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis, Dec, 2017. Accessed: March 14, 2024.

[11] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. page 1093 – 1110, 2017. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85069567720&partnerID=40&md5=97d513457ea0094cd1ad952158d92003. Cited by: 1414.

[12] KrebsOnSecurity. Source code for iot botnet 'mirai' released. https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/, October 1, 2016. Accessed: March 14, 2024.

[13] Fernando Maymí, Robert Bixler, Randolph Jones, and Scott Lathrop. Towards a definition of cyberspace tactics, techniques and procedures. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4674–4679, Dec 2017. doi: 10.1109/BigData.2017.8258514.

[14] Wiem Tounsi and Helmi Rais. A survey on technical threat intelligence in the age of sophisticated cyber attacks. *Computers & Security*, 72:212–233, 2018. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2017.09.001. URL https://www.sciencedirect.com/science/article/pii/S0167404817301839.

[15] Andrei Costin and Jonas Zaddach. Iot malware: Comprehensive survey, analysis framework and case studies. *BlackHat USA*, 1(1):1–9, 2018.

[16] Princy Victor, Arash Habibi Lashkari, Rongxing Lu, Tinshu Sasi, Pulei Xiong, and Shahrear Iqbal. Iot malware: An attribute-based taxonomy, detection mechanisms and challenges. *Peer-to-Peer Networking and Applications*, 16:1–52, 05 2023. doi: 10.1007/s12083-023-01478-w.

[17] Sanjay Madan, Sanjeev Sofat, and Divya Bansal. Tools and techniques for collection and analysis of internet-of-things malware: A systematic state-of-art review. *Journal of King Saud University - Computer and Information Sciences*, 34(10, Part B):9867–9888, 2022. ISSN 1319-1578. doi: https://doi.org/10.1016/j.jksuci.2021.12.016. URL https://www.sciencedirect.com/science/article/pii/S1319157821003621.

[18] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 161–175, 2018. doi: 10.1109/SP.2018.00054.

[19] Ionut Ilascu. Hackers target docker, hadoop, redis, confluence with new golang malware. https://www.bleepingcomputer.com/news/security/hackers-target-docker-hadoop-redis-confluence-with-new-golang-malware/, March 6, 2024. Accessed: July 5, 2024.

[20] Satish Pokhrel, Robert Abbas, and Bhulok Aryal. Iot security: botnet detection in iot using machine learning. *arXiv preprint arXiv:2104.02231*, 2021.

[21] Georgios Kambourakis, Constantinos Kolias, and Angelos Stavrou. The mirai botnet and the iot zombie armies. volume 2017-October, page 267 – 272, 2017. doi: 10.1109/MILCOM.2017.8170867. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85042370568&doi=10.1109%2fMILCOM.2017.8170867&partnerID=40&md5=bbf23ba66c44425a6b0d0d9c6405b5f1. Cited by: 170.

[22] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitras. The dropper effect: Insights into malware distribution with downloader graph analytics. volume 2015-October, page 1118 – 1129, 2015. doi: 10.1145/2810103.2813724. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84954111602&doi=10.1145%2f2810103.2813724&partnerID=40&md5=721a818bb36b53289527016fb893a3a7. Cited by: 91.

[23] Muhammad Junaid Bohio. Analyzing a backdoor/bot for the mips platform. https://www.sans.org/white-papers/35902/, April 13, 2015. Accessed: April 8, 2024.

[24] CrowdStrike. What are wiper attacks? https://www.crowdstrike.com/cybersecurity-101/attack-types/wiper-attack/, December 21, 2023. Accessed: April 8, 2024.

[25] Ioannis Stellios, Panayiotis Kotzanikolaou, Mihalis Psarakis, Cristina Alcaraz, and Javier Lopez. A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services. *IEEE Communications Surveys & Tutorials*, 20(4):3453–3495, 2018. doi: 10.1109/COMST.2018.2855563.

[26] MIVD and AIVD of the Netherlands. Ministry of defence of the netherlands uncovers coathanger,a stealthy chinese fortigate rat. https://www.ncsc.nl/binaries/ncsc/documenten/publicaties/2024/februari/6/mivd-aivd-advisory-coathanger-tlp-clear/TLP-CLEAR+MIVD+AIVD+Advisory+COATHANGER.pdf, February 9, 2024. Accessed: April 8, 2024.

[27] TOM HEGEL JUAN ANDRÉS GUERRERO-SAADE. Acidpour | new embedded wiper variant of acidrain appears in ukraine. https://www.sentinelone.com/labs/acidpour-new-embedded-wiper-variant-of-acidrain-appears-in-ukraine/, March 21, 2024. Accessed: April 8, 2024.

[28] Bryson Lingenfelter, Iman Vakilinia, and Shamik Sengupta. Analyzing variation among iot botnets using medium interaction honeypots. page 761 – 767, 2020. doi: 10.1109/CCWC47524.2020.9031234. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083072475&doi=10.1109%2fCCWC47524.2020.9031234&partnerID=40&md5=850a523654160208a81f7703cfb47647. Cited by: 14.

[29] Daniel Plohmann, Martin Clauß, Steffen Enders, and Elmar Padilla. Malpedia: A collaborative effort to inventorize the malware landscape. 12 2017. doi: 10.18464/cybin.v3i1.17.

[30] Herman Slatman. awesome-threat-intelligence. https://github.com/hslatman/awesome-threat-intelligence?tab=readme-ov-file#sources. Accessed: April 9, 2024.

[31] Tamas K Lengyel. Malware collection and analysis via hardware virtualization. 2015.

[32] Tamas K. Lengyel, Steve Maresca, Bryan D. Payne, George D. Webster, Sebastian Vogl, and Aggelos Kiayias. Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, page 386–395, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330053. doi: 10.1145/2664243.2664252. URL https://doi.org/10.1145/2664243.2664252.

[33] Quoc-Dung Ngo, Huy-Trung Nguyen, Van-Hoang Le, and Doan-Hieu Nguyen. A survey of iot malware and detection methods based on static features. *ICT Express*, 6(4):280–286, 2020. ISSN 2405-9595. doi: https://doi.org/10.1016/j.icte.2020.04.005. URL https://www.sciencedirect.com/science/article/pii/S2405959520300503.

[34] Neil Fox. Yara rules guide: Learning this malware research tool. https://www.varonis.com/blog/yara-rules, May 17, 2021. Updated: June 16, 2023; accessed: March 23, 2024.

[35] VirusTotal. Elf module. https://yara.readthedocs.io/en/latest/modules/elf.html, Updated: Jun 20, 2022. Accessed: July 8, 2024.

[36] VirusTotal. Math module. https://yara.readthedocs.io/en/latest/modules/math.html, Updated: Nov 29, 2023. Accessed: July 8, 2024.

[37] Philip O'Kane, Sakir Sezer, and Kieran McLaughlin. Obfuscation: The hidden malware. *IEEE Security & Privacy*, 9(5):41–47, 2011. doi: 10.1109/MSP.2011.98.

[38] Alexandra Martin. Actionable threat intel (iii) - introducing the definitive yara editor. https://blog.virustotal.com/2023/07/actionable-threat-intel-iii-introducing.html, July 13, 2023. Accessed: March 14, 2024.

[39] Chainguard. bincapz. https://github.com/chainguard-dev/bincapz, Feb 2024. Accessed: July 8, 2024.

[40] Emanuele Cozzi, Pierre-Antoine Vervier, Matteo Dell'Amico, Yun Shen, Leyla Bilge, and Davide Balzarotti. The tangled genealogy of iot malware. In *Proceedings of the 36th Annual Computer Security Applications Conference*, pages 1–16, 2020.

[41] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. In *Technical report*. The MITRE Corporation, 2018.

[42] Black Lotus Labs. Kv-botnet: Don't call it a comeback. https://blog.lumen.com/kv-botnet-dont-call-it-a-comeback/, Feb 7, 2024. Accessed: July 10, 2024.

[43] QiAnXin Threat Intelligence Center. Kaiji botnet resurfaces, unmasking ares hacking group? https://ti.qianxin.com/blog/articles/Kaiji-Botnet-Resurfaces-Unmasking-Ares-Hacking-Group-EN/, Feb 27, 2023. Accessed: July 11, 2024.

[44] NewSky Security. Tracking the people behind botnets: A list of top 20 iot blackhat hackers. https://blog.newskysecurity.com/tracking-the-people-behind-botnets-a-list-of-top-20-iot-blackhat-hackers-3a67d7bd3b Oct 30, 2018. Accessed: July 11, 2024.

[45] Forescout Research Vedere Labs. Department of justice disrupts moobot botnet commandeered by russian apt28: analysis of attacks against routers and malware samples. https://www.forescout.com/blog/doj-moobot-botnet-commandeered-by-russian-apt28-analysis-of-attacks-against-routers Feb 16, 2024. Accessed: July 10, 2024.

[46] Imperva. Google dorking. https://www.imperva.com/learn/application-security/google-dorking-hacking/. Accessed: March 24, 2024.

[47] Mandiant. Unearthing apt44: Russia's notorious cyber sabotage unit sandworm. https://cloud.google.com/blog/topics/threat-intelligence/apt44-unearthing-sandworm, Apr 17, 2024. Accessed: July 17, 2024.

[48] Chang Liu, Rebecca Saul, Yihao Sun, Edward Raff, Maya Fuchs, Townsend Southard Pantano, James Holt, and Kristopher Micinski. Assemblage: Automatic binary dataset construction for machine learning, 2024. URL https://arxiv.org/abs/2405.03991.

[49] NSFOCUS Security Labs. Keksec botnets: Lolfme. https://web.archive.org/web/20210903093833/https://blog.nsfocus.net/keksec-lolfme/, Sep 03, 2021. Accessed: July 22, 2024.

[50] Securonix Threat Labs. Detecting enemybot - securonix initial coverage advisory. https://www.securonix.com/blog/detecting-the-enemybot-botnet-advisory//. Accessed: July 22, 2024.

[51] Forescout Research Vedere Labs. Emerging iot wiper malware: Kaden and new lolfme botnet variants. https://www.forescout.com/blog/emerging-iot-wiper-malware-kaden-and-new-lolfme-botnet/, July 18, 2024. Accessed: July 20, 2024.

[52] Sanseo. Analysis of the rekoobe backdoor being used in attacks against linux systems in korea. https://asec.ahnlab.com/en/55229/, July 11, 2023. Accessed: July 25, 2024.

[53] Felix Aimé. Walking on apt31 infrastructure footprints. https://blog.sekoia.io/walking-on-apt31-infrastructure-footprints/, Nov 10, 2021. Accessed: July 25, 2024.

[54] Huawei. Security notice - statement on remote code execution vulnerability in huawei hg532 product. https://www.huawei.com/en/psirt/security-notices/huawei-sn-20171130-01-hg532-en, Nov 30, 2017. Accessed: July 25, 2024, Updated: Jul 13, 2021.

[55] Nitzan Yaakov Aqua Sec. Muhstik malware targets message queuing services applications. https://www.aquasec.com/blog/muhstik-malware-targets-message-queuing-services-applications/, June 4, 2024. Accessed: July 25, 2024.

[56] Marc-Etienne M.Léveillé ESET Research. Ebury is alive but unseen: 400k linux servers compromised for cryptocurrency theft and financial gain. https://www.welivesecurity.com/en/eset-research/ebury-alive-unseen-400k-linux-servers-compromised-cryptotheft-financial-gain/, May 14, 2024. Accessed: July 25, 2024.

[57] William Ballenthin Mandiant Moritz Raabe. Automatically extracting obfuscated strings from malware using the fireeye labs obfuscated string solver (floss). https://cloud.google.com/blog/topics/threat-intelligence/automatically-extracting-obfuscated-strings/, June 23, 2016. Accessed: July 29, 2024.

[58] William Largent Cisco TALOS. New vpnfilter malware targets at least 500k networking devices worldwide. https://blog.talosintelligence.com/vpnfilter/, May 23, 2018. Accessed: July 29, 2024.

[59] Víctor Manuel Álvarez VirusTotal. Yara is dead, long live yara-x. https://blog.virustotal.com/2024/05/yara-is-dead-long-live-yara-x.html, May 20, 2024. Accessed: July 29, 2024.

Appendices

Appendix A

IoT Malware Knowledge Base

A.1 JSON Format

```
{
    "name": "TheMoon",
    "sources": [
      {
        "name": "Linksys Worm (\"TheMoon\") Captured - SANS Internet Storm Center",
        "link": "https://isc.sans.edu/diary/Linksys+Worm+Captured/17630"
      },
      {
        "name": "Linksys Worm \"TheMoon\" Summary: What we know so far - SANS Internet Storm Center",
        "link": "https://isc.sans.edu/diary/Linksys+Worm+%22TheMoon%22+Summary%3A+What+we+know+so+far/17633"
      },
      {
        "name": "'The Moon' worm infects Linksys routers | Computerworld",
        "link": "https://www.computerworld.com/article/2487791/-the-moon--worm-infects-linksys-routers.html"
      },
      {
        "name": "There's now an exploit for 'TheMoon' worm targeting Linksys routers | Computerworld",
        "link": "https://www.computerworld.com/article/2487778/there-s-now-an-exploit-for--themoon--worm-targeting-linksys-routers.html"
      },
      {
        "name": "35902.pdf (egnyte.com)",
        "link": "https://sansorg.egnyte.com/dl/OaSJMfxJrZ"
      },
      {
        "name": "TheMoon - A P2P botnet targeting Home Routers (fortinet.com)",
        "link": "https://www.fortinet.com/blog/threat-research/themoon-a-p2p-botnet-targeting-home-routers"
      },
      {
        "name": "The Reigning King, Challengers of IP Camera Botnets | Trend Micro (US)",
```

```
      "link": "https://www.trendmicro.com/en_us/research/17/f/reigning-king-ip-camera-botnets-challengers.html"
    },
    {
      "name": "TheMoon-botnet.pdf | Wayback Machine (archive.org)",
      "link": "https://web.archive.org/web/20210123143846/https://blog.netlab.360.com/file/TheMoon-botnet.pdf"
    },
    {
      "name": "TheMoon :An old calendar and a new variant of a botnet (360.com)",
      "link": "https://blog.netlab.360.com/themoon-botnet-a-review-and-new-features/"
    },
    {
      "name": "GPON Exploit in the Wild (IV) - TheMoon Botnet Join in with a 0day(?) (360.com)",
      "link": "https://blog.netlab.360.com/gpon-exploit-in-the-wild-iv-themoon-botnet-join-in-with-a-0day/"
    },
    {
      "name": "TheMoon botnet\u00a0is now leveraging a zero-day to target GPON routers (securityaffairs.com)",
      "link": "https://securityaffairs.com/72762/malware/themoon-gpon-routers.html"
    },
    {
      "name": "TheMoon Rises Again, With a Botnet-as-a-Service Threat | Threatpost",
      "link": "https://threatpost.com/themoon-botnet-as-a-service/141393/"
    },
    {
      "name": "PoC Attack Escalates MikroTik Router Bug to =E2=80=98As Bad As It Gets | Threatpost",
      "link": "https://threatpost.com/poc-attack-escalates-mikrotik-router-bug-to-as-bad-as-it-gets/138076/"
    },
    {
      "name": "A New Phase of TheMoon - Lumen",
      "link": "https://blog.lumen.com/a-new-phase-of-themoon/"
    },
    {
      "name": "The Darkside of TheMoon - Lumen",
      "link": "https://blog.lumen.com/the-darkside-of-themoon/"
    }
  ],
  "resources": [
    {
      "name": "IOCs/Moon_Faceless_IOCs.txt at main =C2=B7 blacklotuslabs/IOCs =C2=B7 GitHub",
      "link": "https://github.com/blacklotuslabs/IOCs/blob/main/Moon_Faceless_IOCs.txt"
    }
  ],
  "type": [
    "worm"
```

```
    ],
    "language": [
      "C"
    ],
    "purpose": [
      "proxy server",
      "propagation"
    ],
    "timeline": [
      "02/2014",
      "06/2016",
      "10/2016",
      "04/2017",
      "05/2018",
      "01/2024",
      "02/2024"
    ],
    "targeted devices": [
      {
        "vendor": "Belkin",
        "models": []
      },
      {
        "vendor": "Linksys",
        "models": [
          "E4200",
          "E3200",
          "E2500",
          "E300",
          "WRT610N",
          "E1000",
          "E1200",
          "E1500",
          "E1550",
          "E2000",
          "E3000",
          "WAG320N",
          "WAP300N",
          "WAP610N",
          "WES610N",
          "WET610N",
          "WRT610N",
          "WRT600N",
          "WRT400N",
          "WRT320N",
          "WRT160N",
          "WRT150N"
        ]
      },
      {
        "vendor": "ASUS",
        "models": [
          "RT-AC66U",
```

```
        "RT-N66U"
      ]
    },
    {
      "vendor": "Tplink",
      "models": [
        "WDR4300"
      ]
    }
  ],
  "initial access": [
    {
      "id": "EDB-ID:31683",
      "source": "exploitdb",
      "link": "https://vulners.com/exploitdb/EDB-ID:31683"
    },
    {
      "id": "CVE-2014-9583",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2014-9583"
    },
    {
      "id": "CVE-2022-28956",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2022-28956"
    },
    {
      "id": "CVE-2021-40655",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2021-40655"
    },
    {
      "id": "CVE-2021-40654",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2021-40654"
    },
    {
      "id": "CVE-2018-10106",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2018-10106"
    },
    {
      "id": "CVE-2022-36755",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2022-36755"
    },
    {
      "id": "CVE-2020-9376",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2020-9376"
    },
    {
      "id": "cve-2020-15894",
```

```
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/cve-2020-15894"
    },
    {
      "id": "CVE-2017-9828",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2017-9828"
    },
    {
      "id": "CVE-2017-9829",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2017-9829"
    },
    {
      "id": "CVE-2018-10561",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2018-10561"
    },
    {
      "id": "CVE-2018-10562",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2018-10562"
    },
    {
      "id": "CVE-2018-14847",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2018-14847"
    },
    {
      "id": "CVE-2018-1156",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2018-1156"
    },
    {
      "id": "CVE-2018-1157",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2018-1157"
    },
    {
      "id": "CVE-2018-1159",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2018-1159"
    },
    {
      "id": "CVE-2018-1158",
      "source": "nist",
      "link": "https://nvd.nist.gov/vuln/detail/CVE-2018-1158"
    }
  ],
  "malpedia": {
    "id": "elf.themoon",
    "name": "TheMoon",
    "link": "https://malpedia.caad.fkie.fraunhofer.de/details/elf.themoon"
```

```
232     }
233   }
```

Appendix B

Behavioral Indicator Extraction - bincapz output

B.1 JSON Format


```
{
        "Path": "wiper/known_wipers/acidpour1",
        "SHA256": "6
            a8824048417abe156a16455b8e29170f8347312894fde2aabe644c4995d7728",
        "Pledge": [
            "stdio"
        ],
        "Behaviors": {
            "evasion/binary/opaque": {
                "Description": "opaque binary",
                "RiskScore": 2,
                "RiskLevel": "MEDIUM",
                "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                    rules/evasion/binary-opaque.yara#opaque_binary"
            },
            "evasion/packer/elf": {
                "Description": "Obfuscated ELF binary (missing content)",
                "RiskScore": 3,
                "RiskLevel": "HIGH",
                "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                    rules/evasion/packer/elf.yara#obfuscated_elf"
            },
            "fs/loopback": {
                "Description": "uses loopback pseudo-device files",
                "MatchStrings": [
                    "/dev/loop"
                ],
                "RiskScore": 1,
                "RiskLevel": "LOW",
                "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                    rules/fs/loopback.yara#loopback"
            },
            "kernel/dev/block/device": {
                "Description": "access raw generic block devices",
                "MatchStrings": [
```

```
                "/dev/sdXX"
            ],
            "RiskScore": 2,
            "RiskLevel": "MEDIUM",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/kernel/dev/block-device.yara#dev_sd"
        },
        "kernel/dev/diskmapper": {
            "Description": "access raw LVM disk mapper devices",
            "MatchStrings": [
                "/dev/dm-XX"
            ],
            "RiskScore": 2,
            "RiskLevel": "MEDIUM",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/kernel/dev/diskmapper.yara#dev_dm"
        },
        "kernel/dev/flash_memory": {
            "Description": "access raw flash memory devices",
            "MatchStrings": [
                "/dev/block/mtdblockXX",
                "/dev/mtdXX",
                "/dev/mtdblockXX"
            ],
            "RiskScore": 2,
            "RiskLevel": "MEDIUM",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/kernel/dev/flash_memory.yara#dev_mtd"
        },
        "kernel/dev/loopback": {
            "Description": "access virtual block devices (loopback)",
            "MatchStrings": [
                "/dev/loopXX"
            ],
            "RiskScore": 2,
            "RiskLevel": "MEDIUM",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/kernel/dev/loopback.yara#dev_loopback"
        },
        "kernel/dev/sd_mmc": {
            "Description": "access raw SD/MMC devices",
            "MatchStrings": [
                "/dev/block/mmcblkXX",
                "/dev/mmcblkXX"
            ],
            "RiskScore": 3,
            "RiskLevel": "HIGH",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/kernel/dev/sd_mmc.yara#dev_mmc"
        },
        "kernel/dev/ubi": {
            "Description": "access raw unsorted block images (UBI)",
            "MatchStrings": [
```

```
                "/dev/ubiXX"
            ],
            "RiskScore": 3,
            "RiskLevel": "HIGH",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/kernel/dev/ubi.yara#ubi"
        },
        "kernel/reboot": {
            "Description": "Forcibly reboots machine",
            "MatchStrings": [
                "/usr/bin/reboot",
                "/usr/sbin/reboot"
            ],
            "RiskScore": 3,
            "RiskLevel": "HIGH",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/kernel/reboot.yara#reboot_command_val"
        },
        "procfs/self/exe": {
            "Description": "gets executable associated to this process",
            "MatchStrings": [
                "/proc/self/exe"
            ],
            "RiskScore": 2,
            "RiskLevel": "MEDIUM",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/procfs/self-exe.yara#proc_self_exe"
        },
        "ref/path/dev/null": {
            "Description": "References /dev/null",
            "MatchStrings": [
                "/dev/null"
            ],
            "RiskScore": 0,
            "RiskLevel": "NONE",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/ref/path/dev-null.yara#dev_null"
        },
        "ref/path/usr": {
            "Description": "path reference within /usr/",
            "MatchStrings": [
                "/usr/bin/reboot",
                "/usr/sbin/reboot"
            ],
            "RiskScore": 0,
            "RiskLevel": "NONE",
            "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                rules/ref/path/usr.yara#usr_path"
        },
        "ref/path/usr/bin": {
            "Description": "path reference within /usr/bin",
            "MatchStrings": [
                "/usr/bin/reboot"
```

```
                ],
                "RiskScore": 1,
                "RiskLevel": "LOW",
                "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                    rules/ref/path/usr-bin.yara#usr_bin_path"
            },
            "ref/path/usr/sbin": {
                "Description": "path reference within /usr/sbin",
                "MatchStrings": [
                    "/usr/sbin/reboot"
                ],
                "RiskScore": 1,
                "RiskLevel": "LOW",
                "RuleURL": "https://github.com/chainguard-dev/bincapz/blob/main/
                    rules/ref/path/usr-sbin.yara#usr_sbin_path"
            }
        },
        "RiskScore": 4,
        "RiskLevel": "CRITICAL"
    }
```

Appendix C

Automated Indicator Selection

C.1 Python Script Output for one Sample

```
apt/v2-final-results/samples/009decf1f36c8a9b081dabf112d0056da1a7b0bed6a30b4b420ef72f3b427bf1:
    $c2_ref {'Remote control'}
    $c2_c_socket {'socket'}
    $c2_c_bind {'bind'}
    $c2_c_listen {'listen'}
    $c2_c_accept {'accept'}
    $c2_c_sockoptions {'getsockopt', 'setsockopt'}
    $c2_c_connect {'connect'}
    $c2_c_recv {'recvfrom', 'recv'}
    $c2_c_send {'send', 'sendto'}
    $c2_c_write {'writev'}
    $c2_ssltls_init {'SSL_SESSION_new', 'SSL_new'}
    $c2_ssltls_accept {'SSL3_ACCEPT', 'DTLS1_ACCEPT', 'SSL2_ACCEPT', 'SSL23_ACCEPT'}
    $c2_ssltls_connect {'DTLS1_CONNECT', 'SSL3_CONNECT', 'SSL23_CONNECT', 'SSL2_CONNECT'}
    $c2_ssltls_read {'SSL3_READ_BYTES', 'DTLS1_READ_BYTES', 'SSL2_READ_INTERNAL', 'SSL3_READ_N'}
    $c2_ssltls_write {'SSL3_WRITE_PENDING', 'SSL3_WRITE_BYTES'}
    $c2_ssltls_version {'TLSv1', 'SSLv2'}
    $c2_proxy_socks_name {'SOCKS5', 'SOCKS4'}
    $c2_proxy_socks_ref {'SOCKS proxy'}
    $c2_proxyref {'Proxy', 'proxy'}
    $c2_tunnelref {'tunnel'}
    $c2_cryptoref {'crypto'}
    $ip_lookuphost {'gethostbyname'}
    $ip_formating {'%d.%d.%d.%d'}
    $ip_parse {'inet_pton', 'inet_addr', 'inet_ntoa'}
    $crypto_ref_key_private {'_PrivateKey_', '_private_key', '_PrivateKey', 'privatekey'}
    $crypto_ref_key_public {'publickey'}
    $crypto_ciphers {'ECDH', 'AES', 'ecdsa', 'RC4', 'ECDSA', 'rc4', 'ecdh'}
    $crypto_ciphers_rc4_1 {'=\x01'}
    $crypto_ciphers_rc4_2 {'\x81ù\x01', '\x81ú\x01', '\x81ÿ\x01', '\x81þ\x01', '\x81û\x01', '\x81ý\x01'}
    $crypto_ciphers_rc4_3 {'H=\x01'}
    $crypto_ciphers_rc4_4 {'H\x81þ\x01', 'H\x81ú\x01', 'H\x81ý\x01', 'H\x81û\x01'}
    $not_evasion_obfuscated {'__gmon_start__', '__libc_start_main', 'dlsym'}
    $evasion_rm {'rm -fR'}
    $evasion_table_xor {'abcdefghijklmnopqrstuvwxyz', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'}
    $run_external_exec {'execv'}
    $run_shell_command {'sh'}
    $run_relative_path {'./libcudart'}
    $file_url_construction {'ftp://', 'https://', 'http://'}
    $file_http_request {'%s %s%s HTTP'}
    $file_use {'fflush(', 'rewind', 'ftell', 'fwrite(', 'fread', 'fread(', 'fwrite', 'fflush', 'setbuf'}
    $file_fopen {'fopen(', 'freopen', 'fopen64', 'fdopen', 'fopen'}
    $file_fetch_commands {'tftp'}
    $file_op_logs {"%s: %sCouldn't write state file: %s"}
    $spread_target_f {'target'}
    $spread_target_p {'pthread', 'srand'}
```

FIGURE C.1: The figure includes the file path containing the SHA256 hash of the corresponding malware sample in the first line, followed by each identified behavioral indicator and the specific evidence from the sample's content that matched it.

Appendix D

AVClass Classifier Tool

D.1 Input


```
{
  "md5": "3f2a65264954c01818f9920d2da76e68",
  "sha1": "47b8a5bee0e2017c2437ed47cda85312d788e1c2",
  "sha256": "992c90213f5af4bb9bc4c14c4ed7924639a11e4639dc6aef165766e312118bf4",
  "av_labels": [
    [
      "ALYac",
      "Trojan.Generic.35951859"
    ],
    [
      "AVG",
      "ELF:Agent-AYQ [Trj]"
    ],
    [
      "Antiy-AVL",
      "Trojan/Linux.Mirai.bzh"
    ],
    [
      "Arcabit",
      "Trojan.Generic.D22494F3"
    ],
    [
      "Avast",
      "ELF:Agent-AYQ [Trj]"
    ],
    [
      "Avast-Mobile",
      "ELF:Gafgyt-KS [Trj]"
    ],
    [
      "Avira",
      "EXP/ELF.Mirai.Z.A"
    ],
    [
      "BitDefender",
      "Trojan.Generic.35951859"
    ],
```

```
    [
      "BitDefenderTheta",
      "Gen:NN.Mirai.36808"
    ],
    [
      "ClamAV",
      "Unix.Trojan.Mirai-8041698-0"
    ],
    [
      "Cynet",
      "Malicious (score: 99)"
    ],
    [
      "DrWeb",
      "Linux.Siggen.9999"
    ],
    [
      "ESET-NOD32",
      "a variant of Linux/Mirai.BZH"
    ],
    [
      "Emsisoft",
      "Trojan.Generic.35951859 (B)"
    ],
    [
      "F-Secure",
      "Exploit.EXP/ELF.Mirai.Z.A"
    ],
    [
      "FireEye",
      "Trojan.Generic.35951859"
    ],
    [
      "Fortinet",
      "ELF/Gafgyt.WN!tr"
    ],
    [
      "GData",
      "Linux.Trojan.Gafgyt.B"
    ],
    [
      "Google",
      "Detected"
    ],
    [
      "Ikarus",
      "Trojan.ELF.Mirai"
    ],
    [
      "Jiangmin",
      "Backdoor.Linux.iuyu"
    ],
    [
```

```
      "K7GW",
      "Trojan ( 0040f06f1 )"
    ],
    [
      "Kaspersky",
      "HEUR:Backdoor.Linux.Mirai.cw"
    ],
    [
      "Kingsoft",
      "elf.Mirai.2002004"
    ],
    [
      "Lionic",
      "Trojan.Linux.Mirai.K!c"
    ],
    [
      "MAX",
      "malware (ai score=99)"
    ],
    [
      "MaxSecure",
      "Trojan.Malware.121218.susgen"
    ],
    [
      "MicroWorld-eScan",
      "Trojan.Generic.35951859"
    ],
    [
      "Microsoft",
      "Backdoor:Linux/Gafgyt.AW!xp"
    ],
    [
      "Rising",
      "Backdoor.Mirai/Linux!1.E816 (CLASSIC)"
    ],
    [
      "Sangfor",
      "Suspicious.Linux.Save.a"
    ],
    [
      "Skyhigh",
      "Artemis!Trojan"
    ],
    [
      "Sophos",
      "Mal/Generic-S"
    ],
    [
      "Symantec",
      "Linux.Mirai"
    ],
    [
      "Tencent",
```

```
      "Backdoor.Linux.Mirai.wao"
    ],
    [
      "TrendMicro",
      "Backdoor.Linux.BASHLITE.SMJC8"
    ],
    [
      "TrendMicro-HouseCall",
      "Backdoor.Linux.BASHLITE.SMJC8"
    ],
    [
      "VIPRE",
      "Trojan.Generic.35951859"
    ],
    [
      "Varist",
      "E32/Mirai.EN.gen!Camelot"
    ],
    [
      "ZoneAlarm",
      "HEUR:Backdoor.Linux.Mirai.cw"
    ],
    [
      "alibabacloud",
      "DDOS:Linux/Mirai"
    ]
  ]
}
```


D.2 Raw output


```
3f2a65264954c01818f9920d2da76e68\t41\tFAM:mirai|15,FILE:os:linux|15,BEH:server|6,
    CLASS:backdoor|6,FAM:gafgyt|5
```


D.3 Processed output


```
{
  "sha256": "992c90213f5af4bb9bc4c14c4ed7924639a11e4639dc6aef165766e312118bf4",
  "detections": "41",
  "families": [
    "mirai",
    "gafgyt"
  ],
  "behaviors": [
    "server"
  ],
  "classes": [
    "backdoor"
  ]
```

```
14 }
```

Appendix E

Datasets of Malware Samples

E.1 Known wiper dataset

Variant	sha256
acidpour	6a8824048417abe156a16455b8e29170f8347312894fde2aabe644c4995d7728
acidrain	9b4dfaca873961174ba935fddaf696145afe7bbf5734509f95feb54f3584fd9a
brickerbot	8bcaf4b826598b45c80d40d1b028cc6a6b6e142f258d4581414f3b2c369a5754
handymannypot	2d94d605bc2753108b548f1fba69a51e7457ee0ef6046705a7c7d7bab960bfb0
handymannypot	5fad60aaf198a99a1f0c0bc166223de70744002174aff822afcca170f906a361
handymannypot	8ef289d261d88b29a7dae2c6b2975ee1c920945ce4b2fb44dec464ec0df4d9d8
handymannypot	b4da8f72ecb5a0b4bfcb7a0cb999bb25ac125aa4431f618b9e6ad3dd0c9fa3d1
handymannypot	c932cd4a230ae880cb67866297b009802b02964b48879135cbd946de08b9f63d
handymannypot	da7f7a47665c6a97b209b85e1301ce9b950a994038236719d2a989443e6a290f
handymannypot	fbab06ea971319bedc944bb06af6c8b042fa47c5016e01325727b536ff628c34
heh	4f9b895a2785f9788fcae8743ab04a24b62e0962b1f8a28dc1206c52327b7916
silex	273bcb899a6d62e828493e8d1bdcdd2776dfef80c96a17cb7d0e2aa37264208a
silex	57c10bfc53873d9a6e3fb15f3973a2ffdc97a7131bdd854fd4c8176f5d85a0ef
silex	7d3aa7438406f0b3fbd2b1d7d3a71e095ddca1efc0fe9939a27e3f51177c1d84
silex	90d262b764e6666bab1aa644177488bbbea3008ddae14ffa669a4c6c16dd3ab9
silex	a3921ef8a4e35579371b7a44b69f2df262802d9b006995666dd9332066ed4aa6
silex	bf318737d708790e74751e7c2238099ceb56270e1a322f91cad51e9e0772b9d9
silex	db77c2d2db676dfe90914e42ef6bdf0c769a9f8fe3468f0379708927aebb63e6
silex	efc45c4cbfa7eab064484a1a9d6ad20b3316d35559007dbf86ea38f2baf84613
silex	f98bcde75347ea723f0b0f70cce6dfe75525d1be9ca776a868da9402f2e06aff
vpnfilter	acf32f21ec3955d6116973b3f1a85f19f237880a80cdf584e29f08bd12666999
vpnfilter	47f521bd6be19f823bfd3a72d851d6f3440a6c4cc3d940190bdc9b6dd53a83d6
vpnfilter	d09f88baf33b901cc8a054d86879b81a81c19be45f8e05484376c213f0eedda2
vpnfilter	2af043730b632d237964dd6abd24a7f6db9dc83aab583532a1238b4d4188396b
vpnfilter	4bfc43761e2ddb65fedab520c6a17cc47c0a06eda33d11664f892fcf08995875

E.2 Known APT deployed malware dataset

Variant	sha256
cyclopsblink	1454338b1bbb692dadb90c758ba8789f56c48dd52f9f94b6dc6784f0944e20f9

cyclopsblink	145bf0e879d544a17364c53e1e695adab8e927fe196cc0d21ad14be3e2cb469f
cyclopsblink	36b3a9dcb283fb0f9fd45f4a371006228d206ec0bdd9e3392eb2d07e72f8d7b0
cyclopsblink	3830213049d64b09f637563faa470b0f2edd0034aa9e92f7908374bd1d6df116
cyclopsblink	4e69bbb61329ace36fbe62f9fb6ca49c37e2e5a5293545c44d155641934e39d1
cyclopsblink	4ec5e0c5dccc5891d39ea76e3c3d3e26d8830d7aa4d63db6084dbfbec6f0d211
cyclopsblink	6f4ee4e05483ca3db54040506ac21a2b49d2bd12379cafad54764907be228556
cyclopsblink	82c3f5092d45ce0e19ac42adaf6632b954b8e78d399f673724956a89c1826d7b
cyclopsblink	88e568afd69fbc944a8d8268e41f2f6100e8bb007083175884ea4149033f4fcf
cyclopsblink	cc3d51578a9dcc7e955061881490e54883904956f5ca5ee2918cd3b249415e59
cyclopsblink	d186f553ad6b38951fdebabfe7ecb4ca6d86ac702a9e8c90a338ad668afdf490
cyclopsblink	fc1e50172c0ce221452b967d1ef705f11bbfe2d54c533d68bd2a7a094605df2d
hipid	3d18bb8b9a5af20ab10441c8cd40feff0aabdd3f4c669ad40111e3aa5e8c54b8
hipid	9603b62268c2bbb06da5c99572c3dc2ec988c49c86db2abc391acf53c1cccceb
hipid	cb1a536e11ae1000c1b29233544377263732ca67cd679f3f6b20016fbd429817
iotreaper	1e0eaaa541bcb16a612f212b2f5a329320db55b003ee617462b9184802a87f9f
iotreaper	2acb0bc56baddeb26a091ff12a39463130243321720d0789375887f4117d8c1a
iotreaper	541f38214b1d9befd5a4173401f34132bb333a97a8d4caf1ababa622d91cc3b6
iotreaper	575de38b34043614bce241610aea7e84cd5fd099b105dc94505f94fc5641b16f
iotreaper	7561ef97bbfa1ebbbe88d2be6a4d40c137b41353516b18ee40bf21eb2058a977
iotreaper	a1cb7e8ceb9d27f5b220509e4a532478fa046f39aabbf7b10cd015f04ac52c85
iotreaper	bb25e66c6da5b01e7538c4ab0c0d56e0449bf0def6f9f272c0732a948c837f2b
iotreaper	c07123bc2eb5702858860af173cd4eab44bd295411b851f725f84ffe37fd43b8
iotreaper	da20e2642cb4d7fa3b99bf2cb88804b44ed48e16f3c68b7c3418208f0d532a10
iotreaper	fbe3e593f37773eacdbfaf7bd9a66d6d7fcbc290f5b2b3068298081841f6ea19
kvbotnet	0279435f8727cca99bee575d157187787174d39f6872c2067de23afc681fe586
kvbotnet	07118af421f14a7e07601639f44a72f6782757ae74d2afffdb531b8209697e7f
kvbotnet	08d0da0c36089f7a1f700b989f2f7825c5ba2549a20735d0bd1e64ca9c4885bc
kvbotnet	19aa5a2235ee2518826a48363cb603060ee73ddccdf7d93bf197f97d7402aa37
kvbotnet	2711f1341d2f150a0c3e2d596939805d66ba7c6403346513d1fc826324f63c87
kvbotnet	2cb6df289475457e807fc202a2b4688b2e23a88c94a8431981780caf8b76acf7
kvbotnet	36c63d0c2a78497ccf555e84f0233a514943faeff38281d99d00baf5df23f184
kvbotnet	3fab16ec4643d8f6b9a99d85427322f7fb40e9ea3cd4de8318c6a52e29869d5a
kvbotnet	48299c2c568ce5f0d4f801b4aee0a6109b68613d2948ce4948334bbd7adc49eb
kvbotnet	5512cce87ff9dfd3ee9721eb29302d1700199ed7d625e09f9f779772ec06bdb0
kvbotnet	5a2681ea2e1d0d5e7db2a2499d2e6e27b2689830c638d5ee28c2eef9867ececf
kvbotnet	6a8230e66011e0a0012273f7d12110c23b1e33bd7232dc67a836662a3d1075c7
kvbotnet	86f01d5342ec39c65b1cff716f19c334cec26a82b87492d783d5e8f4ff9cb63a
kvbotnet	88fc3816c94f9b0191179f4e933843ee4cfdbcb392968605491a387b1235ec12
kvbotnet	8e35d8643c00d9e2993625b03366a7cd1bd36e6a60bc0c6039a509fccf9df150
kvbotnet	9e6a2a01decc2c26f3586a119b6fd3a886c4cf9c76aa452339d164fda40c63e4
kvbotnet	b4f2470159ca93f9d585ae2df1da972f6d14a0c418ebc202a324b9be5c877b61
kvbotnet	b6226c3e0e4ad64bbda3e6a79eb464c7050faa25d1f5332dcac014d2e79dd87f
kvbotnet	b845ef0f9c5853ad1c226ac0ae7bb91159d5bb132185c1bfd171696b755a9164
kvbotnet	bf0ed245e897c7d1ada511db2939e8f3a879a96543f2651d5631339d5419bb75
kvbotnet	c0871ecfe8b306074c6d376db14d966578a8511e5b5d355a4cf2c4d0b8c9deb9
kvbotnet	c524e118b1e263fccac6e94365b3a0b148a53ea96df21c8377ccd8ec3d6a0874
kvbotnet	c71d04e2b6b35fdd058b4be5cf9ea3478697950378d4ee3c7fe0bf87e1e3730f
kvbotnet	d6cd1636569bba4131462bb8f45be1daa9a203aa343b6f2fd48a4847acfc29fa
kvbotnet	e88b03465c0376463f912a5601a518cc697330dc3e5857068f3de0c434b52c9a
kvbotnet	f5271fcb895977dc1eead64415e525323cd412e3f2625aee2fafbb5674beea28

moobot	104e3ea9a190ba039488f5200824fe883b98f6fe01d05a1b55e15ed2199c807a
moobot	17257ce42246b8c47f9ec639a6ffaca2bc14c21a22c4419bf468e3f1d491e330
moobot	28aee94e9a3f6c4296663bb853a5af5817ae109f066c88b7a245316a9a1e4712
moobot	2ae805b68d7408cc40ad058bc0b8b2b5c29d77760084a5230448e47cec1c43f4
moobot	2f182a6cb72712c340c2adb43843cfccb5916d236485de1c62fb40c883570824
moobot	4a932ccc8a45db6897a11de118cdbf67062569112f1caa69793669c5c24be708
moobot	4d35ae9669db428b72b1aaadd21dbed44ad2fc678efc8110d89ff723e0497406
moobot	53d687868fd7ab9e78aa09f696923bd3c057e4e50432d07210080474a8d879cb
moobot	681a00df2e2cc680a4b68bdb6fe7d55c34d6d3fc35d462c78ebb659f9cb2cd60
moobot	844cc1807cc5b628b7aa807ef3b682d051c8ad5427df3d3e36c7e7633bfc5768
moobot	88f2d42bf225c930bc644f82bbd229e170d53dd1072e846e2883265a7ac33301
moobot	ad3fd3eb7a3a276ec0d384afb5b75fe7d9fc047bb0dab40f9d55870d4520c1f3
moobot	bd0ea597f24bb72f8db34b6b6d2c0bc70eb53df9eae40cdb216a13521145ab03
moobot	c290ab5d8ce9fcaa91da3b488c93dee1a4d0581c1335f19cb48027a5a03fe525
moobot	f6541b569787aa050c54ad85976ac5b729697a022be188b0040d37aa91e49ae2
pakdoor	1d60edb577641ce47dc2a8299f8b7f878e37120b192655aaf80d1cde5ee482d2
pakdoor	e1999a3e5a611312e16bb65bb5a880dfedbab8d4d2c0a5d3ed1ed926a3f63e94
ptty	944be9bb167a2f76fe2f539d3860bbf26301830c479bc68509af46e047993c8c
sidewalk	0bff46518b35ddfe37f4a7820286aab829d81f1480d9eeca5aaedc9ceda6724f
sidewalk	a5bad9b8fe8d51400e44646a946f96e33421a0b7ff2bae60b3e0b5621e4fcaf1
sidewalk	be97d7ae3b2d876f027d99d8d61dbca92513f4975336c2ebc26cf8a0839b67b6
speculoos	6943fbb194317d344ca9911b7abb11b684d3dca4c29adcbcff39291822902167
speculoos	99c5dbeb545af3ef1f0f9643449015988c4e02bf8a7164b5d6c86f67e6dc2d28
vpnfilter	00c9bbc56388e3fffc6e53ef846ad269e7e31d631fe6068ff4dc6c09fb40c48b
vpnfilter	01d51b011937433568db646a5fa66e1d25f1321f444319a9fba78fd5efd49445
vpnfilter	0424167da27214cf2be0b04c8855b4cdb969f67998c6b8e719dd45b377e70353
vpnfilter	055bbe33c12a5cdaf50c089a29eaecba2ccf312dfe5e96183b810eb6b95d6c5a
vpnfilter	0649fda8888d701eb2f91e6e0a05a2e2be714f564497c44a3813082ef8ff250b
vpnfilter	081e72d96b750a38ef45e74d0176beb982905af4df6b8654ea81768be2f84497
vpnfilter	099a0b821f77cb4a6e6d4a641ed52ee8fea659ee23b657e6dae75bb8ca3418c3
vpnfilter	0dc1e3f36dc4835db978a3175a462aa96de30df3e5031c5d0d8308cdd60cbede
vpnfilter	0e0094d9bd396a6594da8e21911a3982cd737b445f591581560d766755097d92
vpnfilter	0f3746f273281472e7181f1dd1237f0c9fc26f576a883f42413c759f381006c4
vpnfilter	11533eedc1143a33c1deae105e1b2b2f295c8445e1879567115adebfdda569e2
vpnfilter	116d584de3673994e716e86fbb3945e0c6102bfbd30c48b13872a808091e6bc9
vpnfilter	1367060db50187eca00ad1eb0f4656d3734d1ccea5d2d62f31f21d4f895e0a69
vpnfilter	14984efdd5343c4d51df7c79fd6a2dfd791aa611a751cc5039eb95ba65a18a54
vpnfilter	181408e6ce1a215577c1daa195e0e7dea1fe9b785f9908b4d8e923a2a831fce8
vpnfilter	1cb3b3e652275656b3ae824da5fb330cccd8b27892fb29adc96e5f6132b98517
vpnfilter	1e741ec9452aab85a2f7d8682ef4e553cd74892e629012d903b521b21e3a15bf
vpnfilter	1e824654afba03678f8177e065c487a07192069711eeb4abe397010771b463b5
vpnfilter	1f26b69a353198bb047dde86d48198be8271e07f8c9d647d2f562207e1330a37
vpnfilter	218233cc5ef659df4f5fdabe028ab43bc66451b49a6bfa85a5ed436cfb8dbc32
vpnfilter	24b3931e7d0f65f60bbb49e639b2a4c77de83648ff08e097ff0fa6a53f5c7102
vpnfilter	2aa7bc9961b0478c552daa91976227cfa60c3d4bd8f051e3ca7415ceaeb604ca
vpnfilter	2af043730b632d237964dd6abd24a7f6db9dc83aab583532a1238b4d4188396b
vpnfilter	2b39634dce9e7bb36e338764ef56fd37be6cd0faa07ee3673c6e842115e3ceb1
vpnfilter	2c2412e43f3fd24d766832f0944368d4632c6aa9f5a9610ab39d23e79756e240
vpnfilter	2ef0e5c66f6d46ddef62015ea786b2e2f5a96d94ab9350dd1073d746b6922859
vpnfilter	2ffbe27983bc5c6178b2d447d8121cefaa5ffa87fe7b9e4f68272ce54787492f

vpnfilter	313d29f490619e796057d50ba8f1d4b0b73d4d4c6391cf35baaaace71ea9ac37
vpnfilter	33d6414dcf91b9a665d38faf4ae1f63b7aa4589fe04bdd75999a5e429a53364a
vpnfilter	36e3d47f33269bef3e6dd4d497e93ece85de77258768e2fa611137fa0de9a043
vpnfilter	375ededc5c20af22bdc381115d6a8ce2f80db88a5a92ebaa43c723a3d27fb0d6
vpnfilter	37e29b0ea7a9b97597385a12f525e13c3a7d02ba4161a6946f2a7d978cc045b4
vpnfilter	39dc1aded01daaf01890db56880f665d6cafab3dea0ac523a48aa6d6e6346fff
vpnfilter	3bbdf7019ed35412ce4b10b7621faf42acf604f91e5ee8a903eb58bde15688ff
vpnfilter	3bd34426641b149c40263e94dca5610a9ecfcbce69bfdd145dff1b5008402314
vpnfilter	3df17f01c4850b96b00e90c880fdfabbd11c64a8707d24488485dd12fae8ec85
vpnfilter	4497af1407d33faa7b41de0c4d0741df439d2e44df1437d8e583737a07ec04a1
vpnfilter	47f521bd6be19f823bfd3a72d851d6f3440a6c4cc3d940190bdc9b6dd53a83d6
vpnfilter	4896f0e4bc104f49901c07bc84791c04ad1003d5d265ab7d99fd5f40ec0b327f
vpnfilter	48bfcbc3162a0b00412cba5eff6c0376e1ae4cfbd6e35c9ea92d2ab961c90342
vpnfilter	49a0e5951dbb1685aaa1a6d2acf362cbf735a786334ca131f6f78a4e4c018ed9
vpnfilter	4af2f66d7704de6ff017253825801c95f76c28f51f49ee70746896df307cbc29
vpnfilter	4b03288e9e44d214426a02327223b5e516b1ea29ce72fa25a2fcef9aa65c4b0b
vpnfilter	4beba775f0e0b757ff32ee86782bf42e997b11b90d5a30e5d65b45662363ece2
vpnfilter	4bfc43761e2ddb65fedab520c6a17cc47c0a06eda33d11664f892fcf08995875
vpnfilter	4c596877fa7bb7ca49fb78036b85f92b581d8f41c5bc1fa38476da9647987416
vpnfilter	4c5e21125738c330af1bfe5cabc5f18fa14bbef53805dda2c3c31974555f7ec5
vpnfilter	4c8da690501c0073a3c262a3079d8efac3fea9e2db9c55f3c512589e9364e85c
vpnfilter	4cbf9ecb6ca4f2efed86ba6ebf49436c65afe7ae523ec9dae58e432a9d9a89d0
vpnfilter	4d6cbde39a81f2c62d112118945b5eeb1d73479386c962ed3b03d775e0dccfa0
vpnfilter	4e022e4e4ee28ae475921c49763ee620b53bf11c2ad5fffe018ad09c3cb078cc
vpnfilter	4ffe074ad2365dfb13c1c9ce14a5e635b19acb34a636bae16faf9449fb4a0687
vpnfilter	50ac4fcd3fbc8abcaa766449841b3a0a684b3e217fc40935f1ac22c34c58a9ec
vpnfilter	51e92ba8dac0f93fc755cb98979d066234260eafc7654088c5be320f431a34fa
vpnfilter	579b2e6290c1f7340795e42d57ba300f96aef035886e80f80cd5d0bb4626b5fc
vpnfilter	5a28ad479d55275452e892b799c32803f81307079777bb1a5c4d24477206d16b
vpnfilter	5be57b589e5601683218bb89787463ca47ce3b283d8751820d30eee5e231678c
vpnfilter	5cf43c433fa1e253e937224254a63dc7e5ad6c4b3ab7a66ec9db76a268b4deeb
vpnfilter	5d94d2b5f856e5a1fc3a3315d3cd03940384103481584b80e9d95e29431f5f7a
vpnfilter	5dabbce674b797aaa42052b501fb42b20be74d9ffcb0995d933fbf786c438178
vpnfilter	5e715754e9da9ed972050513b4566fb922cd87958ecf472d1d14cd76923ae59a
vpnfilter	5f6ee521311e166243d3e65d0253d12d1506750c80cd21f6a195be519b5d697f
vpnfilter	638957e2def5a8fda7e3efefff286e1a81280d520d5f8f23e037c5d74c62553c
vpnfilter	6449aaf6a8153a9ccbcef2e2738f1e81c0d06227f5cf4823a6d113568f305d2a
vpnfilter	66a98ad0256681313053c46375cb5c144c81bf4b206aaa57332eb5f1f7176b8c
vpnfilter	6807497869d9b4101c335b1688782ab545b0f4526c1e7dd5782c9deb52ee3df4
vpnfilter	6a76e3e98775b1d86b037b5ee291ccfcffb5a98f66319175f4b54b6c36d2f2bf
vpnfilter	6d8877b17795bb0c69352da59ce8a6bfd7257da30bd0370eed8428fad54f3128
vpnfilter	6e7bbf25ea4e83229f6fa6b2fa0f880dde1594a7bec2aac02ff7d2d19945d036
vpnfilter	6eb09f805a68b29c9516d649019bea0bb4796e504ca379783455508a08f61087
vpnfilter	7093cc81f32c8ce5e138a4af08de6515380f4f23ed470b89e6613bee361159e1
vpnfilter	70c271f37dc8c3af22fdcad96d326fe3c71b911a82da31a992c05da1042ac06d
vpnfilter	776cb9a7a9f5afbaffdd4dbd052c6420030b2c7c3058c1455e0a79df0e6f7a1d
vpnfilter	78fee8982625d125f17cf802d9b597605d02e5ea431e903f7537964883cf5714
vpnfilter	797e31c6c34448fbecda10385e9ccfa7239bb823ac8e33a4a7fd1671a89fe0f6
vpnfilter	7a66d65fa69b857beeeaaef67ec835900eee09a350b6f51f51c83919c9223793
vpnfilter	7ba0dc46510492a7f6c9b2bcc155333898d677cd8a88fe0e1ac1ad3852f1c170

vpnfilter	7e5dca90985a9fac8f115eaacd8e198d1b06367e929597a3decd452aaa99864b
vpnfilter	7ee215469a7886486a62fea8fa62d3907f59cf9bf5486a5fe3a0da96dabea3f9
vpnfilter	80c20db74c54554d9936a627939c3c7ea44316e7670e2f7f5231c0db23bc2114
vpnfilter	81cbe57cd80b752386ee707b86f075ad9ab4b3a97f951d118835f0f96b3ae79d
vpnfilter	830091904dab92467956b91555bc88fa7e6bbde514b8a90bb078c8a3bb2f39a9
vpnfilter	83b3dbf7f6bc5f98151b26781fa892fc1a014c62af18c95ae537848204f413b8
vpnfilter	840ba484395e15782f436a7b2e1eec2d4bf5847dfd5d4787ae64f3a5f668ed4f
vpnfilter	84227f906c7f49071d6598b9035fc785d2b144a6349d0cf7c29177c00db2dc2f
vpnfilter	8440128350e98375b7eff67a147dfe4e85067d67f2ad20d9485f3de246505a5f
vpnfilter	8505ece4360faf3f454e5b47239f28c48d61c719b521e4e728bc12d951ecf315
vpnfilter	879be2fa5a50b7239b398d1809e2758c727e584784ba456d8b113fc98b6315a2
vpnfilter	8a20dc9538d639623878a3d3d18d88da8b635ea52e5e2d0c2cce4a8c5a703db1
vpnfilter	8de0f244d507b25370394ba158bd4c03a7f24c6627e42d9418fb992a06eb29d8
vpnfilter	90efcaeac13ef87620bcaaf2260a12895675c74d0820000b3cd152057125d802
vpnfilter	94eefb8cf1388e431de95cab6402caa788846b523d493cf8c3a1aa025d6b4809
vpnfilter	952f46c5618bf53305d22e0eae4be1be79329a78ad7ec34232f2708209b2517c
vpnfilter	95840bd9a508ce6889d29b61084ec00649c9a19d44a29aedc86e2c34f30c8baf
vpnfilter	9683b04123d7e9fe4c8c26c69b09c2233f7e1440f828837422ce330040782d17
vpnfilter	97d00fc2bc5f5c9a56b498cf83b7a801e2c11c056772c5308ee7adea50556309
vpnfilter	98112bd4710e6ffe389a2beb13ff1162017f62a1255c492f29238626e99509f3
vpnfilter	99944ad90c7b35fb6721e2e249b76b3e8412e7f35f6f95d7fd3a5969eaa99f3d
vpnfilter	9b455619b4cbfeb6496c1246ba9ce0e4ffa6736fd536a0f99686c7e185eb2e22
vpnfilter	9e854d40f22675a0f1534f7c31626fd3b67d5799f8eea4bd2e2d4be187d9e1c7
vpnfilter	9eb6c779dbad1b717caa462d8e040852759436ed79cc2172692339bc62432387
vpnfilter	a15b871fcb31c032b0e0661a2d3dd39664fa2d7982ff0dbc0796f3e9893aed9a
vpnfilter	a168d561665221f992f51829e0b282eeb213b8aca3a9735dbbaecc4d699f66b9
vpnfilter	a3cf96b65f624c755b46a68e8f50532571cee74b3c6f7e34eecb514a1eb400cf
vpnfilter	a41da0945ca5b5f56d5a868d64763b3a085b7017e3568e6d49834f11952cb927
vpnfilter	a43a4a218cf5755ce7a7744702bb45a34321339ab673863bf6f00ac193cf55fc
vpnfilter	a6e3831b07ab88f45df9ffac0c34c4452c76541c2acd215de8d0109a32968ace
vpnfilter	aa5baa135b2ada5560833747260545d6a5b49558f6244c0f19443dc87c00294d
vpnfilter	ab789a5a10b4c4cd7a0eb92bbfcf2cc50cb53066838a02cfb56a76417de379c5
vpnfilter	acf32f21ec3955d6116973b3f1a85f19f237880a80cdf584e29f08bd12666999
vpnfilter	acfc72b8d6611dc9cd6a3f1a4484aa0adfb404ad5faaa8b8db5747b0ff05bc22
vpnfilter	ae1353e8efe25b277f52decfab2d656541ffdf7fd10466d3a734658f1bc1187a
vpnfilter	afacb38ea3a3cafe0f8dbd26dee7de3d0b24cdecae280a9b884fbad5ed195de7
vpnfilter	afd281639e26a717aead65b1886f98d6d6c258736016023b4e59de30b7348719
vpnfilter	b0edf66d4f07e5f58b082f5b8479d48fbab3dbe70eba0d7e8254c8d3a5e852ef
vpnfilter	b118b23a192f372616efe8c2b12977d379ac76df22493c14361587bd1cc8a804
vpnfilter	b2dd77af9dd9e8d7d4ebc778f00ff01c53b860a04c4e0b497f2ae74bb8a280c0
vpnfilter	b301d6f2ba8e532b6e219f3d9608a56d643b8f289cfe96d61ab898b4eab0e3f5
vpnfilter	b431aebc2783e72be84af351e9536e8110000c53ebb5db25e89021dc1a83625e
vpnfilter	b9770ec366271dacdae8f5088218f65a6c0dd82553dd93f41ede586353986124
vpnfilter	ba9fee47dcc7bad8a7473405aabf587e5c8d396d5dd5f6f8f90f0ff48cc6a9ce
vpnfilter	bc51836048158373e2b2f3cdb98dc3028290e8180a4e460129fef0d96133ea2e
vpnfilter	be3ddd71a54ec947ba873e3e10f140f807e1ae362fd087d402eff67f6f955467
vpnfilter	bfd028f78b546eda12c0d5d13f70ab27dff32b04df3291fd46814f486ba13693
vpnfilter	c084c20c94dbbffed76d911629796744eff9f96d24529b0af1e78cda54cdbf02
vpnfilter	c0cfb87a8faed76a41f39a4b0a35ac6847ffc6ae2235af998ee1b575e055fac2
vpnfilter	c2bcde93227eb1c150e555e4590156fe59929d3b8534a0e2c5f3b21ede02afa0

vpnfilter	c8a82876beed822226192ea3fe01e3bd1bb0838ab13b24c3a6926bce6d84411b
vpnfilter	ca0bb6a819506801fa4805d07ee2ebaa5c29e6f5973148fe25ed6d75089c06a7
vpnfilter	cccbf9bff47b3fd391274d322076847a3254c95f95266ef06a3ca8be75549a4b
vpnfilter	d09f88baf33b901cc8a054d86879b81a81c19be45f8e05484376c213f0eedda2
vpnfilter	d1bc07b962ccc6e3596aa238bb7eda13003ea3ca95be27e8244e485165642548
vpnfilter	d2de662480783072b82dd4d52ab6c57911a1e84806c229f614b26306d5981d98
vpnfilter	d6097e942dd0fdc1fb28ec1814780e6ecc169ec6d24f9954e71954eedbc4c70e
vpnfilter	d9a60a47e142ddd61f6c3324f302b35feeca684a71c09657ddb4901a715bd4c5
vpnfilter	dbede977518143bcee6044ed86b8178c6fc9d454fa346c089523eedee637f3be
vpnfilter	dd88273437031498b485c380968f282d09c9bd2373ef569952bc7496ebadadde
vpnfilter	e6c5437e8a23d50d44ee47ad6e7ce67081e7926a034d2ac4c848f98102ddb2f8
vpnfilter	e70a8e8b0cd3c59cca8a886caa8b60efb652058f50cc9ff73a90bc55c0dc0866
vpnfilter	e74ae353b68a1d0f64b9c8306b2db46dfc760c1d91bfdf05483042d422bff572
vpnfilter	e75e224c909c9ead4cb50cd772f606407b09b146051bfb28015fcbe27b4a5e8d
vpnfilter	eaf879370387a99e6339377a6149e289655236acc8de88324462dcd0f22383ff
vpnfilter	ec88fe46732d9aa6ba53eed99e4d116b7444afd2a52db988ea82f883f6d30268
vpnfilter	eeb3981771e448b7b9536ba5d7cd70330402328a884443a899696a661e4e64e5
vpnfilter	f30a0fe494a871bd7d117d41025e8d2e17cd545131e6f27d59b5e65e7ab50d92
vpnfilter	f3d0759dfab3fbf8b6511a4d8b5fc087273a63cbb96517f0583c2cce3ff788b8
vpnfilter	f4f0117d2784a3b8dfef4b5cb7f2583dd4100c32f9ee020f16402508e073f0a1
vpnfilter	f5d06c52fe4ddca0ebc35fddbbc1f3a406bdaa5527ca831153b74f51c9f9d1b0
vpnfilter	f989df3aeede247a29a1f85fc478155b9613d4a416428188eda1a21bd481713a
vpnfilter	fa229cd78c343a7811cf8314febbc355bb9baab05b270e58a3e5d47b68a7fc7d
vpnfilter	fa4b286eeaf7d74fe8f3fb36d80746e18d2a7f4c034ae6c3fa4c917646a9e147
vpnfilter	fc9594611445de4a0ba30daf60a7e4dec442b2e5d25685e92a875aca2c0112c9
vpnfilter	fcb6ff6a679ca17d9b36a543b08c42c6d06014d11002c09ba7c38b405b50debe
vpnfilter	fce03f57b3fd3842efac3ce676687794c4decc29b612068e578134f3c4c4296a
vpnfilter	fe46a19803108381d2e8b5653cc5dce1581a234f91c555bbfff63b289b81a3dc
vpnfilter	fe9c17ac036622b2d73466f62b5d095edda2d3b60fa546a48d0bb18f8b11059f
vpnfilter	ff118edb9312c85b0b7ff4af1fc48eb1d8c7c8da3c0e1205c398d2fe4a795f4b
vpnfilter	ff471a98342bafbab0d341e0db0b3b9569f806d0988a5de0d8560b6729875b3e
vpnfilter	ff70462cb3fc6ddd061fbd775bbc824569f1c09425877174d43f08be360b2b58
vpnfilter	ffb0e244e0dabbaabf7fedd878923b9b30b487b3e60f4a2cf7c0d7509b6963ba

Appendix F

Vulnerability Table

Vulnerability Name	Products	Vulnerability ID	Exploit-DB ID
UPnP SOAP TelnetD Command Execution	D-Link	CVE-2013-7471	28333, 27044
Eir WAN Side Remote Command Injection	Eir D1000	CVE-2016-10372	40740
Netgear setup.cgi unauthenticated RCE	Netgear DGN1000	CVE-2016-11059	43055, 25978
HNAP SoapAction-Header Command Execution	D-Link	CVE-2015-2051, CVE-2019-10892	37171, 38722
JAWS Webserver Unauthenticated Shell Command Execution	MVPower	CVE-2016-20016	41471
Netgear cgi-bin Command Injection	Netgear R6250, R6400, R6700, R6900, R7000, R7100LG, R7300DST, R7900, R8000, D6220, D6400	CVE-2016-6277	41598
Chimay Red' Stack Clash Remote Code Execution	Mikrotik RouterOS	CVE-2017-20149	44283, 44284
D-Link 'command.php' Remote Command Execution	D-Link DIR-600, DIR-300, DIR-610	CVE-2020-9377	27528
NETGEAR ReadyNAS Surveillance Remote Command Execution	NETGEAR ReadyNAS	CVE-2017-18378	42956
D-Link Unauthenticated RCE	D-Link DIR-850	CVE-2024-0769, CVE-2018-10957, CVE-2023-48842	
Linksys 'apply.cgi' Remote Command Injection	Linksys E1000/E1500/E2500	CVE-2013-2679	24936

AVTECH IP Camera / NVR / DVR Devices - Multiple Vulnerabilities	AVTECH IP Camera / NVR / DVR	CVE-2013-4980, CVE-2013-4981, CVE-2013-4982, EDB-ID-40500	40500
D-Link Unauthenticated Remote Access	D-Link DIR-645	CVE-2022-28956, CVE-2021-40655, CVE-2021-40654, CVE-2018-10106, CVE-2022-36755, CVE-2020-9376, CVE-2020-15894	
SAPIDO RB-1732 - Remote Command Execution	SAPIDO RB-1732	CVE-2021-4242	47031
Dbltek GoIP - Local File Inclusion	DBLTek GHSFVT-1.1-67-5 firmware	CVE-2017-16934	50775
NetGain 'ping' Command Injection	NetGain Enterprise Manager 7.2.562	CVE-2017-16608	41499
NUUOS OS Command Injection	NUUO NVRmini 2 3.0.8	CVE-2016-5676, CVE-2018-1149	40212, 40200
D-Link OS Command Injection	D-Link DSL-2750B	CVE-2016-20017	44760
Zyxel Unauthenticated LAN RCE	Zyxel	CVE-2023-28769	
ZyXEL / Billion / TrueOnline RCE	Zyxel P660HN-T, Billion 5200W-T	CVE-2017-18371, CVE-2017-18374, CVE-2017-18370, CVE-2017-18368, CVE-2017-18369, CVE-2017-18372, CVE-2017-18373	43884
Netgear Prosafe RCE	Netgear Prosafe WC9500, WC7600, WC7520	CVE-2016-11022	38097
Linksys Remote Debug Root Shell	Linksys WAP54G	CVE-2010-2506, CVE-2010-2261, CVE-2010-1573	
Netlink GPON RCE	Netlink GPON 1.0.11, Guangzhou 1GE ONU V2801RW, V2804RGW, Parks Fiberlink 210	CVE-2023-33617, CVE-2020-8958, CVE-2022-30023	48225

Symantec Web Gateway XSS / CSRF / SQL Injection / Command Injection	Symantec Web Gateway	CVE-2013-1616, CVE-2013-4670, CVE-2013-4672, CVE-2013-1617, CVE-2013-4671	27136
WAVLINK RCE	WAVLINK WN572HP3, WN533A8, WN530H4, WN535G3, WN531P3	CVE-2022-35526	
HiSilicon DVR/NVR - Remote Backdoor Account	HiSilicon DVR/NVR hi3520d firmware	CVE-2020-22253	48004
ThinkPHP RCE	ThinkPHP 5.0.23/5.1.31	CVE-2019-9082, CVE-2018-20062	45978
VIVOTEK Network Cameras	VIVOTEK Network Cameras	CVE-2017-9829, CVE-2017-9828	
MikrotikOS	MikrotikOS	CVE-2018-14847, CVE-2018-1156, CVE-2018-1157, CVE-2018-1159, CVE-2018-1158	
GPON RCE	GPON	CVE-2018-10561, CVE-2018-10562	
ZHONE Router Web RCE	ZHONE Router Web	CVE-2014-8357, CVE-2014-8356, CVE-2014-9118	38453
TOTOLINK Backdoor / RCE	TOTOLINK	CVE-2015-9550	37770
Seagate BlackArmor NAS RCE	Seagate BlackArmor NAS	CVE-2014-3206	33159
Vacron NVR RCE	Vacron NVR	CVE-2008-4873	6864
EnGenius EnShare RCE	EnGenius EnShare IoT Gigabit Cloud Service	EDB-ID-42114	42114
Liferay Portal - Java Unmarshalling via JSONWS RCE	Liferay Portal	CVE-2020-7961	48332
VestaCP - 'v_sftp_licence' Command Injection	VestaCP 0.9.8	CVE-2021-46850	49674
Realtek SDK - Miniigd UPnP SOAP Command Execution	Realtek SDK	CVE-2014-8361	37169

Linksys RCE	Linksys E-series	EDB-ID-31683	31683
WePresent Command Injection	WePresent WiPG-1000	EDB-ID-41935	41935
Dell KACE Remote Code Execution	Dell KACE Systems Management Appliance (K1000)	CVE-2019-20504	46684
HooToo TripMate Remote Code Execution	Hootoo HT-05	CVE-2018-20841	46143
Belkin Wemo UPnP Remote Code Execution	Belkin Wemo	CVE-2019-12780	46436
ASUS DSL Modem Remote Code Execution	ASUS DSL N12E	CVE-2018-15887	45135
NUUO NVRmini - 'upgrade_handle.php' Remote Command Execution	NUUO NVR	CVE-2018-14933	45070
GoAhead / Wireless IP Camera (P2P) WIFICAM - Remote Code Execution	Wireless IP Camera (P2P) WIFICAM	CVE-2017-8221, CVE-2017-8222, CVE-2017-8223, CVE-2017-8224, CVE-2017-8225, CVE-2017-18377	43142
AVCON6 systems management platform - OGNL RCE	AVCON6 systems management platform	EDB-ID-47379	47379
Sar2HTML 3.2.1 RCE	Sar2HTML 3.2.1	EDB-ID-47204	47204
ACTi ASOC 2200 Web Configurator 2.6 RCE	ACTi ASOC 2200	EDB-ID-16993	16993
3Com Office Connect RCE	3Com OfficeConnect	EDB-ID-9862	9862
CCBill Remote Code Execution	CCBILL CGI	EDB-ID-53	53
Fritz!Box Webcam - Command Injection	Fritz!Box 7270, 7570, 7490, 7390, 7360, 7340, 7330, 7272, 7270, 7170 Annex A A/CH, 7170 Annex B English, 7170 Annex A English, 7140, 7113, 6840 LTE, 6810 LTE, 6360 Cable, 6320 Cable, 5124, 5113, 3390, 3370, 3272, 3270	CVE-2014-9727	32753, 33136

Comcast Xfinity Gateway RCE	Xfinity Gateway, Arris TG1682G	EDB-ID-40856, CVE-2017-16836	40856, 38657
BEWARD IP Camera - RCE	BEWARD N100 H.264 VGA IP Camera M2.1.6	EDB-ID-46320, EDB-ID-46319	46320, 46319
FLIR Thermal Camera FC-S/PT - Command Injection	FLIR FC-Series S FC-334-NTSC, PT-Series PT-334 200562	EDB-ID-42788	42788
EyeLock nano NXT 3.5 - RCE	EyeLock nano NXT	EDB-ID-40228	40228
Iris ID IrisAccess ICU 7000-2 - XSS	Iris ID IrisAccess ICU 7000-2	EDB-ID-40165	40165
SpreeCommerce RCE	Spreecommerce API searchlogic	EDB-ID-17199	17199
op5 7.1.9 - RCE	op5 Monitor	EDB-ID-39676	39676
Mitel AWC - Command Execution	Mitel Audio and Web Conferencing	EDB-ID-15807	15807
Gitorious - Arbitrary Command Execution	Gitorious	EDB-ID-18393	18393
Dogfood CRM - spell.php RCE	Dogfood CRM	EDB-ID-16917	16917
Wansview NCS601W / Belkin F7D7601 - RCE	Wansview NCS601W, Belkin F7D7601	EDB-ID-42331, CVE-2020-35714	42331
Dahua Generation 2/3 - Backdoor Access	Dahua	EDB-ID-44002	44002
Netcore 53413 Backdoor	Netcore, Netis	EDB-ID-43387, CVE-2018-25069	43387
ZTE RCE	ZTE ZXV10 H108L	EDB-ID-38409	38409
CCTV/DVR RCE	CCTVs, DVRs from over 70 vendors	EDB-ID-39596	39596
Hadoop YARN ResourceManager - Command Execution	Hadoop YARN ResourceManager	EDB-ID-45025	45025
OptiLink ONT1GEW GPON RCE	OptiLink ONT1GEW	CVE-2022-40005, EDB-ID-49955	49955
SonicWall Visualdoor RCE	SonicWall SSL-VPN	EDB-ID-49499	49499
MiniDVBLinux 5.4 Remote Root Command Injection	MiniDVBLinux	EDB-ID-51096	51096

Merit/iCatchInc/LILIN DVR/NVR - RCE	LILIN	VU-26285, VU-26286, VU-26287	
NUUO handle_iscsi.php OS Command Injection	NUUO NVRmini 2 NE-4160, NT-4040, NT-4040(R)	EDB-ID-40212	40212
DLink diagnostic.php Command Execution	DLink DIR-645, DIR-815	EDB-ID-24956, CVE-2014-100005	24956
HomeMatic Zentrale CCU2 Unauthenticated RCE	HomeMatic Zentrale CCU2	EDB-ID-45052	45052
ASUS Multi Model CSRF	ASUS RT-N56U, RT-AC66U	CVE-2013-3093	
RUIJIE Command Injection	RUIJIE NBR700	CVE-2021-46226	
KGUARD DVR Vulnerability / DLink RCE	KGUARD DVR, DLink DI-7200GV2.E1	CVE-2015-4464	
Axis SSI RCE	Axis	EDB-ID-43984	43984
TVT OEM API RCE	Shenzhen TVT DVR/NVR/IPC API	SSVID-97217	
TP-Link http/tftp backdoor	TP-Link WDR4300	TP-Link backdoor - 2013	
Edimax RCE	Edimax EW-7438RPn	CVE-2021-35395	48377
Buffalo RCE	Buffalo WSR-2533DHPL2, WSR-2533DHP3	CVE-2021-20090	
TUTOS 1.3 'cmd.php - RCE	TUTOS	CVE-2008-0149, CVE-2008-0148	4861
WordPress Plugin DZS-VideoGallery - XSS / Command Injection	WordPress Plugin DZS-VideoGallery	CVE-2014-9094	39250
Ubiquiti airOS Arbitrary File Upload	Ubiquiti airOS	CVE-2015-9266	39853
WordPress Plugin Slider REvolution 3.0.95 / Showbiz Pro 1.7.1 - Arbitrary File Upload	WordPress Plugin Slider REvolution, Showbiz Pro	CVE-2015-9499	35385, 36957
Magento eCommerce RCE	Magento Shoplift	CVE-2015-1397	37977
Wordpress Plugin Complete Gallery Manager/Woocommerce RCE	Wordpress Plugin Complete Gallery Manager	CVE-2013-5962	28377

WordPress Robo Gallery 2.0.14 RCE	WordPress Robo Gallery	PACKETSTORM-136657	
WordPress Squirrel Theme 1.6.4 RFI	WordPress Squirrel Theme	PACKETSTORM-134688	
WordPress Plugin Gwolle Guestbook 1.5.3 RFI/RCE	WordPress Plugin Gwolle Guestbook	CVE-2015-8351	
WordPress Plugin Site Import 1.0.1 LFI	WordPress Plugin Site Import	EDB-ID-39558	39558
WordPress Plugin Brandfolder 3.0 LFI/RFI	WordPress Plugin Brandfolder	EDB-ID-39591	39591
WordPress Plugin Issuu Panel 1.6 RFI/LFI	WordPress Plugin Issuu Panel	PACKETSTORM-136398	
DLink ShareCenter Backdoor	DLink DNS-320L ShareCenter	EDB-ID-43434	43434

Appendix G

YARA Rules

G.1 Wiper YARA Rule

```
import "vt"
import "elf"

rule wiper {
    meta:
        description = "Identifies wiper IoT malware type."
    strings:
        // Hard-coded device paths:
        // - Accesses raw generic block devices
        $device_hd = /\/dev\/hd[\$%\w\{\}]{0,10}/
        $device_sd = /\/dev\/sd[\$%\w\{\}]{0,10}/
        // - Accesses raw SD/MMC devices
        $device_mmc = /\/dev\/mmcblk[\$%\w\{\}]{0,16}/
    $device_block_mmc = /\/dev\/block\/mmcblk[\$%\w\{\}]{0,16}/
        // - Accesses raw flash memory devices
        $device_mtd = /\/dev\/mtd[\$%\w\{\}]{0,16}/
    $device_block_mtd = /\/dev\/block\/mtdblock[\$%\w\{\}]{0,16}/
        // - Accesses raw LVM disk mapper devices
        $device_dm = /\/dev\/dm-[\$%\w\{\}]{0,10}/
        // - Accesses raw unsorted block images (UBI)
        $device_ubi = /\/dev\/ubi[\$%\w\{\}]{0,16}/
        // - Accesses raw system memory
        $device_mem = "/dev/mem" fullword
        // - Accesses virtual system memory
        $device_kmem = "/dev/kmem" fullword
        // - Accesses ramdisk
        $device_ram = /\/dev\/ram([1]?[0-9]|disk)/


        // Hard-coded malicious data paths:
        // - /dev/zero
        $data_zero = "/dev/zero" fullword
        // - /dev/urandom
        $data_urandom = "/dev/urandom" fullword

        // Malicious commands:
        // - mtd specific
```

```
        $destruct_flash = /(rfdformat|ftl_format|flash_erase(all)?)\s+\/dev\/\w+/
        // - fdisk
        $destruct_fdisk = /fdisk(\s+-[CHS]\s+\d+){1,3}\s+\/dev\/\w+/
        // - cp
        $destruct_cp = /cp\s+(-\w+)?\s*\/dev\/(zero|urandom)\s+[\/\-\w\.\*]+/
        // - cat garbage value redirected to files
        $destruct_cat = /cat\s+\/dev\/(zero|urandom)\s*\>\s*[\/\-\w\.\*]+/
        // - dd
        $destruct_dd = /dd\s+if=\/dev\/(zero|urandom)[\w\%\/\.\-\" \=]{0,64}/
        // - rm
        $destruct_rm = /rm\s*(-[rRfF]{1,2}|--recursive|--force)\s*(-[rRfF]{1,2}|--
            recursive|--force)?\s+\/(\*|dev|\s)/
        // - truncate
        $destruct_truncate = /(ftruncate|truncate|ftruncate64)\s+\-s/
        // - echo
        $destruct_echo = /echo\s+\-n\s+>/

        // Special rm rf command:
        $rm_rf_star = /rm(\s*|-(r|f|rf|fr)|--(recursive|force)){2,9}\/\*/

        // Accesses mounted devices:
        $access_etc_fstab = "/etc/fstab" fullword
    $access_etc_mtab = "/etc/mtab" fullword
        $access_proc_mounts = "/proc/mounts" fullword
        $access_proc_fd = /\/proc\/[%{$][\w\}]{0,12}\/fd/
        $access_fdisk = /fdisk(\s+\-\w+)*\s+\-(l|x|\-list|\-list\-details)/

        // Final commands:
        // - halt
        $final_halt = /(halt\b|halt[^(ed|s)])(\s*\-(n|f|p|\-do\-not\-sync|\-force|\-
            poweroff|\-reboot)){0,7}/
        // - reboot
        $final_reboot = /(reboot\b|reboot[^(ed|s)])(\s*|$)(\s*\-(n|f|p|\-do\-not\-sync
            |\-force|\-poweroff|\-reboot)){0,7}/
        // - poweroff
        $final_poweroff = /poweroff(\s+|$)(\s*\-(n|f|p|\-do\-not\-sync|\-force|\-
            poweroff|\-reboot)){0,7}/
        // - systemctl
        $final_systemctl = /systemctl\s+(reboot|halt|poweroff|kexec)/
        // - shutdown
        $final_shutdown = /shutdown(\s*|-k|-h|-r|-H|-P){1,7}(now|\d+){1}/
        // - init / telinit
        $final_init_telinit = /(init|telinit)\s+(-\w+|\-\-verbose|-e\s+\w+=[\w\-]+)?\s
            *\d+/

        // Obfuscated ELF binary (missing content) from bincapz/rules/evasion/packer/
            elf.yara
        $obfuscated_not_dlsym = "dlsym" fullword
        $obfuscated_not_gcc = "gcc" fullword
        $obfuscated_not_libstdc = "libstdc" fullword
        $obfuscated_not_glibc = "glibc" fullword
        $obfuscated_not_setsid = "setsid" fullword
        $obfuscated_not_gmon = "__gmon_start__"
```

```
        $obfuscated_not_glibc2 = "@GLIBC"
        $obfuscated_not_cxa = "__cxa_finalize"
        $obfuscated_not_dereg = "__deregister_frame_info"
        $obfuscated_not_symtab = ".symtab" fullword
        $obfuscated_not___libc_start_main = "__libc_start_main"

    condition:
        (vt.metadata.exiftool["FileType"] == "ELF executable"
        or vt.metadata.exiftool["FileType"] == "bash script"
        or vt.metadata.exiftool["FileType"] == "sh script"
        or vt.metadata.exiftool["FileType"] == "PHP"
        or (vt.metadata.magic contains "ELF" and vt.metadata.magic contains "
            executable")
        or (vt.metadata.magic contains "shell script" and vt.metadata.magic contains "
            executable")
        or vt.metadata.magic contains "PHP script"
        or (vt.metadata.magic contains "Python script" and vt.metadata.magic contains
            "executable" and vt.metadata.exiftool["Newlines"] == "Unix LF")
        or vt.metadata.magic contains "ASCII C program"
        or elf.type == elf.ET_EXEC)
        and
        vt.metadata.file_size < 6MB
        and
        (
            // First condition = obfuscation + 5 device paths + 1 final command
            (elf.type == elf.ET_EXEC and uint32(0) == 1179403647 and none of (
                $obfuscated_not_*) and 5 of ($device_*) and 1 of ($final_*))
            or
            // Second condition = no obfuscation + 5 device paths + 2 final commands
            (elf.type == elf.ET_EXEC and not (uint32(0) == 1179403647 and none of (
                $obfuscated_not_*)) and 5 of ($device_*) and 2 of ($final_*))
            or
            // Third condition = 3 device paths + 1 query for accessing device paths +
                 1 use of malicious data + 3 destructive commands + 2 final commands
            (3 of ($device_*) and 1 of ($access_*) and 1 of ($data_*) and 3 of (
                $destruct_*) and 2 of ($final_*))
            or
            // Fourth condition = 1 device path + 1 query for accessing device paths +
                 1 rm -rf /* command
            (1 of ($device_*) and 1 of ($access_*) and $rm_rf_star)
        )
}
```

G.2 Generic IoT Malware YARA Rule

```
import "vt"
import "elf"

rule apt {
    meta:
```

```
6       description = "Identifies IoT malware based on behavioral indicators of
            malware families / variants deployed by APTs"
7   strings:
8       // Reference to command and control server
9       $c2_ref = /\b(cnc|c2_|(command|remote)[\-_\s&]*control)\b/i
10      $backdoor_ref = /b[a4]ckd[o0]{2}r/i
11      // ------------------------------------------------------------------------
12
13      // C sockets
14      // - open a socket (client/server)
15      $c2_c_socket = /(__|__libc_|__GI_|\x00)socket(\.c|\.h|@|\x00)/
16      // - bind to, listen on a socket and accept (server)
17      $c2_c_bind = /(__|__libc_|__GI_|\x00)bind(\.c|\.h|@|\x00)/
18      $c2_c_listen = /(__|__libc_|__GI_|\x00)listen(\.c|\.h|@|\x00)/
19      $c2_log_listen = /%s: listen/
20      $c2_c_accept = /(__|__libc_|__GI_|\x00)accept(4|64)?(\.c|\.h|@|\x00)/
21      // - get/set socket options
22      $c2_c_sockoptions = /(__|__libc_|__GI_|\x00)(set|get)sockopt(\.c|\.h|@|\x00)/
23      // - initiate a connection on a socket (client)
24      $c2_c_connect = /(__|__libc_|__GI_|\x00)connect(\.c|\.h|@|\x00)/
25      // - receive a message from a socket
26      $c2_c_recv = /(__|__libc_|__GI_|\x00)(_)?recv(msg|from)?(\.c|\.h|@|\x00)/
27      $c2_log_recv = /%s: recv/
28      // - read data from a socket
29      $c2_c_read = /(__|__libc_|__GI_|\x00)(p)?read(v(2)?)?(\.c|\.h|@|\x00)/
30      $c2_log_read = /%s: read:/
31      // - send a message to a socket
32      $c2_c_send = /(__|__libc_|__GI_|\x00)(_)?send((m)?msg|to)?(\.c|\.h|@|\x00)/
33      $c2_log_send = /%s: sendmsg([\(\)%d]+)?:/
34      // - write data to a socket
35      $c2_c_write = /(__|__libc_|__GI_|\x00)(p)?write(v(2)?)?(\.c|\.h|@|\x00)/
36      $c2_log_write = /%s: write/
37      // - kernel command used by hackers
38      $c2_c_hacker = /(__|__libc_|__GI_|\x00)socketcall(\.c|\.h|@|\x00)/
39      // ------------------------------------------------------------------------
40
41      // BIO sockets
42      $c2_bio_socket = /(BIO_)(s_|new_)?sock(et|_init)/i
43      $c2_bio_bind = /(BIO_)bind/i
44      $c2_bio_listen = /(BIO_)listen/i
45      $c2_bio_accept = /(BIO_)accept(_ex|_new)?/i
46      $c2_bio_connect = /(BIO_)connect(_new)?/i
47      $c2_bio_read = /(BIO_)((n|zlib_)?read(0)?(_ex|_intern)?|get(s|_line))/i
48      $c2_bio_write = /(BIO_)((n|zlib_)?write(0)?(_ex|_intern)?|puts)/i
49      // ------------------------------------------------------------------------
50
51      // SSL/TLS socket communication
52      $c2_ssltls_init = /\x00(WOLFSSL|SSL|(D)?TLS|mbedtls_(net|ssl))[1-3]{0,2}(
            _session)?_(init|new)\x00/i
53      $c2_ssltls_bind = /\x00(WOLFSSL|SSL|(D)?TLS|mbedtls_(net|ssl))[1-3]{0,2}_bind\
            x00/i
54      $c2_ssltls_accept = /\x00(WOLFSSL|SSL|(D)?TLS|mbedtls_(net|ssl))[1-3]{0,2}
            _accept\x00/i
```

```
55        $c2_ssltls_connect = /\x00(WOLFSSL|SSL|(D)?TLS|mbedtls_(net|ssl))[1-3]{0,2}
              _connect(_cert)?\x00/i
56        $c2_ssltls_read = /\x00(WOLFSSL|SSL|(D)?TLS|mbedtls_(net|ssl))[1-3]{0,2}_(read
              |peek|recv)(_internal|_n|_bytes|_ex|_early_data)\x00/i
57        $c2_ssltls_write = /\x00(WOLFSSL|SSL|(D)?TLS|mbedtls_(net|ssl))[1-3]{0,2}_(
              write|send)(v|_internal|_n|_bytes|_pending|_ex|_early_data)\x00/i
58        $c2_ssltls_version = /\x00((WOLFSSL|SSL|(D)?TLS)(v)?[1-3]{1,2}|CyaSSL:|
              PolarSSL(1|test))\x00/
59        // ------------------------------------------------------------------------
60
61        // Proxies and tunneling techniques
62        // - mentions socks4, socks5, microsocks, proxy
63        $c2_proxy_socks_name = /\b(\.)?socks(v\s)?(4|5|\s*%[sd])+\b/i
64        $c2_proxy_socks_ref = /\bsocks[ _\-45]?proxy\b/i
65        $c2_proxyref = /\bproxy\b/i
66        $c2_proxy_microsocks = "microsocks" fullword
67        // - use of the TOR/.onion protocol
68        $c2_proxy_tor = /(\bTor %s\b|HiddenServicePort|\x00(ntor)+\x00|orport=[0-9]* (
              v3ident|id)=[A-Z0-9]*)/i
69        // - tunneling (HTTP, ICMP, IPIP, IPSec, DNS, DNS-over-HTTPS)
70        $c2_tunnel_http = /connect %s http/i
71        $c2_tunnel_ipsec = /ipsec(\s)?tunnel/i
72        $c2_tunnel_ipip= /ipip(\s)?tunnel/i
73        $c2_tunnelref = /\btunnel\b/i
74        // - crypto ref
75        $c2_cryptoref = /\bcrypto\b/i
76        // ------------------------------------------------------------------------
77
78        // IP related
79        // - resolve network host name to IP address
80        $ip_lookuphost = "gethostbyname"
81        // - looks up the reverse hostname for an IP
82        $ip_lookupip = /(\.in-addr\.arpa|ip6\.arpa)/
83        // - string format IP address
84        $ip_formating = /%[du]\.%[du]\.%[du]\.%[du]/
85        // - interacts with the iptables/nftables firewall
86        $ip_iptables = /(iptables|nftables)/ fullword
87        // - parses IP address
88        $ip_parse = /(inet_addr|inet_pton|inet_nto[ap])/
89        // - reuse TCP/IP ports for listening and connecting
90        $ip_reuse_so_readdr = "SO_REUSEADDR"
91        $ip_reuse_so_report = "SO_REUSEPORT"
92        // ------------------------------------------------------------------------
93
94        // Crypto
95        // - certificates
96        $crypto_key_universal = /\-\-\-\-\-BEGIN %s\-\-\-\-\-/i
97        $crypto_key_pem = /\-\-\-\-\-BEGIN (CERTIFICATE|SIGNATURE|DH PARAMETERS|
              CROSSCERT|ED25519 CERT)\-\-\-\-\-/i
98        $crypto_key_private = /\-\-\-\-\-BEGIN (RSA |DH |ED25519 |DSA |EC |ENCRYPTED |
              OPENSSH )?PRIVATE KEY\-\-\-\-\-/i
99        $crypto_key_public = /\-\-\-\-\-BEGIN (RSA |DH |ED25519 |DSA |EC |ENCRYPTED |
              OPENSSH )?PUBLIC KEY\-\-\-\-\-/i
```

```
        // - refs
        $crypto_ref_key_private = /(\b|_)private(_)?key(_|\b)/i
        $crypto_ref_key_public = /(\b|_)public(_)?key(_|\b)/i
        $crypto_ref_key_client_crt = /client(_ca)?\.(crt|key)/i
        $crypto_ref_key_secret = /(\b|_)secret(_)?key(_|\b)/i
        // - SSH public key
        $crypto_ssh_public_key = /ssh-[dr]sa [\w\+\/\=]*/
        // - base64
        $crypto_base64 = /\bbase64(_encod(e|ing)|_decod(e|ing)|\.c|\.h)/i
        // - ciphers
        $crypto_ciphers = /\b(aes|ecdsa|ecdh|rc4|fernet|GOST89|blake2)\b/i
        $crypto_ciphers_rc4_1 = { 3d 00 01 00 00 }
        $crypto_ciphers_rc4_2 = { 81 f? 00 01 00 00 }
        $crypto_ciphers_rc4_3 = { 48 3d 00 01 00 00 }
        $crypto_ciphers_rc4_4 = { 48 81 f? 00 01 00 00 }
        // ------------------------------------------------------------------------

        // Evasion techniques
        // -- Higher scored
        // - Obfuscated ELF binary (missing content) => not $not_evasion_obfuscated
        $not_evasion_obfuscated = /(\x00dlsym\x00|\x00gcc\x00|\x00libstdc\x00|\
            x00glibc\x00|\x00setsid\x00|__gmon_start__|@GLIBC|__cxa_finalize|
            __deregister_frame_info|\x00.symtab\x00|__libc_start_main)/
        // - Pretends to be a kworker // MSIE
        $evasion_pretend = /\b(kworker|compatible; MSIE)\b/i
        // - XOR'ed user agent, often found in backdoors
        $evasion_xor_Mozilla_5_0 = "Mozilla/5.0" ascii wide xor(1-255)
        // -- Lower scored
        // - built by archaic gcc version
        $evasion_gcc_v345 = /GCC: \(.{1,64}\) [345]/i
        // - kill and remove
        $evasion_rm = /rm -[rf]{1,2}/i
        $evasion_kill = /(killall|pgrep|pkill)/
        // - URLs/commands obfuscated using xor
        $evasion_xor_http = "http:" xor(1-31)
        $evasion_xor_https = "https:" xor(1-31)
        $evasion_xor_ftp = "ftp:/" xor(1-31)
        $evasion_xor_office = "office" xor(1-31)
        $evasion_xor_google = "google." xor(1-31)
        $evasion_xor_microsoft = "microsoft" xor(1-31)
        $evasion_xor_apple = "apple." xor(1-31)
        $evasion_xor_user_agent = "User-Agent" xor(1-31)
        $evasion_xor_http2 = "http://" xor(33-255)
        $evasion_xor_https2 = "https://" xor(33-255)
        $evasion_xor_ftp2 = "ftp://" xor(33-255)
        $evasion_xor_google2 = "google." xor(33-255)
        $evasion_xor_microsoft2 = "microsoft" xor(33-255)
        $evasion_xor_apple2 = "apple." xor(33-255)
        $evasion_xor_user_agent2 = "User-Agent" xor(33-255)
        $evasion_xor_b_chmod = "chmod " xor(1-31)
        $evasion_xor_b_curl = "curl -" xor(1-31)
        $evasion_xor_b_bin_sh = "/bin/sh" xor(1-31)
        $evasion_xor_b_bin_bash = "/bin/bash" xor(1-31)
```

```
        $evasion_xor_b_openssl = "openssl" xor(1-31)
        $evasion_xor_b_dev_null = "/dev/null" xor(1-31)
        $evasion_xor_b_usr_bin = "/usr/bin" xor(1-31)
        $evasion_xor_b_usr_sbin = "/usr/sbin" xor(1-31)
        $evasion_xor_b_var_tmp = "/var/tmp" xor(1-31)
        $evasion_xor_b_var_run = "/var/run" xor(1-31)
        $evasion_xor_b_screen_dm = "screen -" xor(1-31)
        $evasion_xor_b_zmodload = "zmodload" xor(1-31)
        $evasion_xor_b_dev_tcp = "/dev/tcp" xor(1-31)
        $evasion_xor_b_bash_i = "bash -i" xor(1-31)
        $evasion_xor_b_bash_c = "bash -c" xor(1-31)
        $evasion_xor_b_base64 = "base64" xor(1-31)
        $evasion_xor_b_eval = "eval(" xor(1-31)
        $evasion_xor_b_chmod2 = "chmod " xor(33-255)
        $evasion_xor_b_curl2 = "curl -" xor(33-255)
        $evasion_xor_b_bin_sh2 = "/bin/sh" xor(33-255)
        $evasion_xor_b_bin_bash2 = "/bin/bash" xor(33-255)
        $evasion_xor_b_openssl2 = "openssl" xor(33-255)
        $evasion_xor_b_dev_null2 = "/dev/null" xor(33-255)
        $evasion_xor_b_usr_bin2 = "/usr/bin" xor(33-255)
        $evasion_xor_b_usr_sbin2 = "/usr/sbin" xor(33-255)
        $evasion_xor_b_var_tmp2 = "/var/tmp" xor(33-255)
        $evasion_xor_b_var_run2 = "/var/run" xor(33-255)
        $evasion_xor_b_screen_dm2 = "screen -" xor(33-255)
        $evasion_xor_b_zmodload2 = "zmodload" xor(33-255)
        $evasion_xor_b_dev_tcp2 = "/dev/tcp" xor(33-255)
        $evasion_xor_b_bash_i2 = "bash -i" xor(33-255)
        $evasion_xor_b_bash_c2 = "bash -c" xor(33-255)
        $evasion_xor_b_base642 = "base64" xor(33-255)
        $evasion_xor_b_eval2 = "eval(" xor(33-255)
        // - hides shell command history
        $evasion_hide_history = /(HIDE_THIS|HISTFILE=(\/dev)?|HISTCONTROL=\"*
            ignorespace|shopt -[ou]{1,2} history|set +o history|history -[cd]|HISTSIZE
            =0)/
        // - contains a table that may be used for XOR decryption
        $evasion_table_xor = /ABCDEFGHIJKLMNOPQRSTUVWXYZ/i
        // - references a non-standard libc library
        $evasion_fake_lib = /(libnetresolv\.so|\blibs\.so\b|libc\.so\.[2-57-9])/
        // ------------------------------------------------------------------------

        // Shell or program execution
        // - executes external programs
        $run_external_exec = /(__|__libc_|__GI_|\x00)((f)?exec[lv]?[pPe]?|posix_spawn)
            (\.c|\.h|@|\x00)/
        $run_external_command = /(__|__libc_|__GI_|\x00)(system|openpty|popen|pipe)(\.
            c|\.h|@|\x00)/
        // - executes shell => any of them
        $run_shell_command = /(\/bin\/|\x00)(ba|da|z)?sh(\s)?/
        // - relative hidden launcher
        $run_relative_hidden = /\.\/\.[a-zA-Z0-9]{1,32}([^.][a-zA-Z0-9]+|\s)/
        // - executes relative path
        $run_relative_path = /\x00\.\/[a-zA-Z0-9 %]{3,32}/
        // - Runs shell commands but throws output away
```

```
        $run_throw_output = /(((1|2|&)?>|-o|--output)\s?\/dev\/null|[12&]>%s)/
        // ---------------------------------------------------------------

        // Persistence
        // - access zsh/bash startup files
        $persist_bash = /(\.|\/etc\/)(bash_profile|profile|bashrc|bash_logout|zprofile
            |zshrc|zlogout)(\s|\b|\x00)/
        // - use crontab to persist => any of ($persist_cron_*) and none of (
            $not_persist_cron_*)
        $persist_cron = /(\/etc(\/config)?\/cron[\/\w\.]{0,32}|crontab -|\/var\/spool
            \/cron|\/var\/cron\/tabs|(\x00|\s).{0,10} \* \* .{0,10}(\x00|\s)|@(reboot|
            daily|midnight|hourly|weekly))/
        // - contains ssh related
        $persist_ssh_binary = /usage: ssh(d)?\s\[/i
        $persist_ssh_path = /\/usr\/(s)?bin\/ssh(d)?/
        $persist_ssh_ref = /\x00ssh(d)?[_\s ]?(tty|connection|client|server)\x00/i
        $persist_ssh_config = /[\$\%\{\}\w\/]{0,16}\.ssh[\w\/]{0,16}/
        $persist_ssh_authorized_hosts = /[\/\.\$\%]{1,32}authorized_keys/
        $persist_ssh_communication = /ssh_msg_(send|recv)/
        // ---------------------------------------------------------------

        // Dropping files
        // - url construction
        $file_url_construction = /([st]?ftp|http(s)?)
            :\/\/([1-9][0-9]{0,2}(\.[0-9]{1,3}){3}(:[1-9][0-9]{0,5})?|[%sd:@]+|\x00)/
        // - http request
        $file_http_request = /(%s|get) [%\/\?=a-z]{2,32} http/i
        // - access file descriptors
        $file_use = /\x00((_){0,2}(IO|GI|stdio|libc|syscall)?(_new)?(_)?(file)?_|
            openssl_|%s: )?(fwrite|fread|fflush|rewind|fgetpos|fsetpos|setbuf|ftell(o)
            ?)(\x00|: %s|\(|64|@|\.[ch])/
        $file_open = /\x00((_){0,2}(IO|GI|stdio|libc|syscall)?(_new)?(_)?(file)?_|
            openssl_|%s: )?(open(at|_2)?|creat)(_nocancel)?(\x00|: %s|\(|64|@|\.[ch])/
        $file_fopen = /\x00((_){0,2}(IO|GI|stdio)?(_new)?(_)?(file)?_|openssl_|%s: )?f
            (d|re)?open(\x00|: %s|\(|64|@|\.[ch]|_)/
        // - invokes download commands
        $file_fetch_commands = /(\x00|\s+)(curl|wget|(t|s)?ftp(get)?)( -| http|\x00)/
        // - invokes chmod
        $file_chmod = /chmod\s+([+rwx]{2,4}|[75]{3,4})/
        // - invokes chattr
        $file_chattr = "chattr" fullword
        // - transfer data between file descriptors (TODO: socket + file descriptor =>
             exfiltration or dropping file)
        $file_sendfile_ref = "sendfile" fullword
        // - logs file operations
        $file_op_logs = /%s:.{0,32}file: %s/
        // ---------------------------------------------------------------

        // Spreading
        // - nmap => $spread_nmap and none of ($not_spread_nmap*)
        $spread_nmap = "nmap" fullword
        // - raw sockets with multiple targets, possible DoS or security scanning tool
```

```
242         $spread_target_r = /(raw socket|HDRINCL|IPPacket|IPPROTO_RAW|SOCK_RAW|\bICMP\b
                )/i
243         $spread_target_f = /(flood|target|attack)/i
244         $spread_target_p = /(pthread|\b(s)?rand\b)/
245         // - list network interfaces => any of them
246         $spread_int_list= /(getifaddrs|freeifaddrs|ifconfig|\/proc\/net\/dev)/
                fullword
247         // - access logs of logins
248         $spread_logins_lastlog = /\/var\/log\/(lastlog|(u|w)tmp(x)?)/ fullword
249         // ----------------------------------------------------------------
250     condition:
251         // Conditions regarding TYPE
252         (vt.metadata.magic contains "ASCII C program" or elf.type == elf.ET_EXEC)
253         and
254         // Conditions regarding SIZE
255         filesize < 5325KB
256         and
257         // Conditions based on behavioral indicators from different APT deployed
                malware families/variants
258         (
259             // First condition from Cyclopsblink
260             (
261                 // 4 C socket comm. behaviors / 4 bio socket comm. behaviors
262                 (4 of ($c2_c_*) or 4 of ($c2_bio_*))
263                 and
264                 // ref. to tunneling with proxy and crypto + 1 evasion
265                 $c2_proxyref and 1 of ($c2_tunnel*) and $c2_cryptoref and 1 of (
                        $evasion_*)
266             )
267             or
268             // Second condition from Hipid / Speculoos
269             (
270                 // without obfuscation + 4 C socket comm. + 1 evasion + 3 run + 1
                        persist + 3 file + 2 spread
271                 ($not_evasion_obfuscated and 4 of ($c2_c_*) and 1 of ($evasion_*) and 3
                         of ($run_*) and 3 of ($file_*) and 2 of ($spread_*))
272                 or
273                 // obfuscation + 1 evasion + 1 run + 1 persist + 1 file + 2 spread
274                 (not $not_evasion_obfuscated and 2 of ($evasion_*) and 1 of ($run_*)
                        and 1 of ($persist_*) and 1 of ($file_*) and 2 of ($spread_*))
275             )
276             or
277             // Third condition from IoTReaper
278             (
279                 // without obfuscation
280                 $not_evasion_obfuscated
281                 and
282                 (
283                     // 4 C socket comm. + 2 evasion + 2 ip
284                     (4 of ($c2_c_*) and 2 of ($evasion_*) and 2 of ($ip_*))
285                     or
286                     // 7 C socket comm. + 1 evasion + 1 ip + 1 run
```

```
287                 (7 of ($c2_c_*) and 1 of ($evasion_*) and 1 of ($ip_*) and 1 of (
                        $run_*))
288             )
289             or
290             // obfuscation + 1 C socket comm. + 5 evasion + 1 run
291             (not $not_evasion_obfuscated and 1 of ($c2_c_*) and 5 of ($evasion_*)
                    and 1 of ($run_*))
292         )
293         or
294         // Fourth condition from KVBotnet
295         (
296             // obfuscation
297             not $not_evasion_obfuscated
298             and
299             (
300                 // 3 evasion + 3 file + 1 run
301                 (3 of ($evasion_*) and 3 of ($file_*) and 1 of ($run_*))
302                 or
303                 // 2 evasion + pretend + file fetch command
304                 (2 of ($evasion_*) and $evasion_pretend and $file_fetch_commands)
305                 or
306                 (
307                     (
308                         // 2 C socket comm.
309                         2 of ($c2_c_*)
310                         or
311                         // 1 C socket comm. + 1 C log comm.
312                         (1 of ($c2_c_*) and 1 of ($c2_log_*))
313                     )
314                     and
315                     (
316                         // 3 evasion + 2 ip + 1 run
317                         (3 of ($evasion_*) and 2 of ($ip_*) and 1 of ($run_*))
318                         or
319                         // 2 evasion + 2 ip + 1 crypto
320                         (2 of ($evasion_*) and 2 of ($ip_*) and 1 of ($crypto_*))
321                         or
322                         // 2 proxy + 2 evasion + 3 ip + 1 run
323                         (2 of ($c2_proxy*) and 2 of ($evasion_*) and 3 of ($ip_*)
                                and 1 of ($run_*))
324                     )
325                 )
326             )
327         )
328         or
329         // Fifth condition from MooBot
330         (
331             // without obfuscation
332             $not_evasion_obfuscated
333             and
334             (
335                 // 8 C socket comm.
336                 8 of ($c2_c_*)
```

```
                or
                // 4 C log comm.
                4 of ($c2_log_*)
            )
            // 2 ip + 2 file + 2 spread
            and 2 of ($ip_*) and 2 of ($file_*) and 2 of ($spread_*) and
            (
                // 3 proxy + 1 run
                (3 of ($c2_proxy*) and 1 of ($run_*))
                or
                // 1 proxy + 1 tunnel ref + 1 crypto ref + 2 evasion + 2 persisit
                (1 of ($c2_proxy*) and $c2_tunnelref and $c2_cryptoref and 2 of (
                    $evasion_*) and 2 of ($persist_*))
            )
        )
        or
        // Sixth condition from Sidewalk
        (
            // without obfuscation
            $not_evasion_obfuscated
            and
            (
                (
                    // 2 C socket comm. + 2 proxy socks + 3 ip
                    2 of ($c2_c_*) and 2 of ($c2_proxy_*) and 3 of ($ip_*)
                    and
                    // ref. to tunneling with proxy and crypto
                    $c2_proxyref and $c2_tunnelref and $c2_cryptoref
                )
                or
                // 4 C socket comm. + 1 ip
                (4 of ($c2_c_*) and 1 of ($ip_*))
            )
            // 4 SSL/TLS comm.
            and 4 of ($c2_ssltls_*)
            // 3 evasion
            and 3 of ($evasion_*)
            // 2 run
            and 2 of ($run_*)
        )
        or
        // Seventh condition from Pakdoor
        (
            // without obfuscation + 1 ip + 1 crypto ref + 1 crypto chipers + 1
                evasion + 1 run + 1 persist + 2 file + 2 spread
            ($not_evasion_obfuscated and 1 of ($ip_*) and 1 of ($crypto_ref_*) and
                1 of ($crypto_ciphers*) and 1 of ($evasion_*) and 1 of ($run_*) and
                 1 of ($persist_*) and 2 of ($file_*) and 2 of ($spread_*))
            or
            // obfuscation + c2/backdoor ref + 2 ip + 1 crypto key + 2 evasion + 1
                run
            (not $not_evasion_obfuscated and ($c2_ref or $backdoor_ref) and 2 of (
                $ip_*) and 1 of ($crypto_key_*) and 2 of ($evasion_*) and 1 of (
```

```
                $run_*))
        )
        or
        // Eight condition from VPNFilter:
        (
            // without obfuscation + 2 evasion + 4 file + 3 spread
            ($not_evasion_obfuscated and 2 of ($evasion_*) and 4 of ($file_*) and 3
                 of ($spread_*))
            or
            (
                // obfuscation
                not $not_evasion_obfuscated
                and
                (
                    // 1 proxy + 3 ip + 1 crypto cipher + 1 evasion + 3 run + 1
                       persist + 3 spread'
                    (1 of ($c2_proxy*) and 3 of ($ip_*) and 1 of ($crypto_ciphers*)
                         and 1 of ($evasion_*) and 3 of ($run_*) and 1 of ($persist_
                       *) and 3 of ($spread_*))
                    or
                    // 1 proxy + 1 evasion + 1 run + 4 file + 2 spread
                    (1 of ($c2_proxy*) and 1 of ($evasion_*) and 1 of ($run_*) and
                       4 of ($file_*) and 2 of ($spread_*))
                    or
                    // 3 proxy + 2 ip + 1 crypto cipher + 2 evasion + 1 run + 1
                       file
                    (1 of ($c2_proxy*) and 2 of ($ip_*) and 1 of ($crypto_ciphers*)
                         and 2 of ($evasion_*) and 1 of ($run_*) and 1 of ($file_*))
                    or
                    // 3 proxy + 2 ip + 1 crypto cipher + 1 evasion + 1 run + 3
                       file
                    (1 of ($c2_proxy*) and 2 of ($ip_*) and 1 of ($crypto_ciphers*)
                         and 1 of ($evasion_*) and 1 of ($run_*) and 3 of ($file_*))
                    or
                    // 2 ip + 1 crypto cipher + 1 evasion + 1 run + 2 file
                    (2 of ($ip_*) and 1 of ($crypto_ciphers*) and 1 of ($evasion_*)
                         and 1 of ($run_*) and 2 of ($file_*))
                    or
                    // 2 ip + 1 crypto cipher + 2 evasion + 1 run + 1 file
                    (2 of ($ip_*) and 1 of ($crypto_ciphers*) and 2 of ($evasion_*)
                         and 1 of ($run_*) and 1 of ($file_*))
                    or
                    // 1 ip + 1 evasion + 1 run + 2 file + 2 spread
                    (1 of ($ip_*) and 1 of ($evasion_*) and 1 of ($run_*) and 2 of
                       ($file_*) and 2 of ($spread_*))
                    or
                    // 3 ip + 1 run + 2 file + 1 spread
                    (3 of ($ip_*) and 1 of ($run_*) and 2 of ($file_*) and 1 of (
                       $spread_*))
                    or
                    // 1 ip + 5 crypto + 1 crypto cipher + 2 evasion + 1 file
                    (1 of ($ip_*) and 4 of ($crypto*) and 1 of ($crypto_ciphers*)
                       and 2 of ($evasion_*) and 1 of ($file_*))
```

```
                    or
                    // 2 crypto + 2 evasion + 1 run + 2 file + 2 spread
                    (2 of ($crypto*) and 2 of ($evasion_*) and 1 of ($run_*) and 2
                        of ($file_*) and 2 of ($spread_*))
                    or
                    // 1 evasion + 1 run + 2 file + 3 spread
                    (1 of ($evasion_*) and 1 of ($run_*) and 2 of ($file_*) and 3
                        of ($spread_*))
                )
            )
        )
    )
}
```