Toward a Labeled Dataset of IoT Malware Features

Stian Hagbø Olsen
Florida Institute of Technology
Melbourne, FL, USA
shagboeolsen2017@my.fit.edu

TJ OConnor
Florida Institute of Technology
Melbourne, FL, USA
toconnor@fit.edu

***Abstract*—IoT malware has accompanied the rapid growth of embedded devices over the last decade. Previous work has proposed static and dynamic detection and classification techniques for IoT malware. However, this work requires a diverse and fine-grained set of malware-specific characteristics. This paper presents a longitudinal, diverse, and open-source IoT malware dataset. To demonstrate the depth of the dataset, we propose an approach for recovering symbol tables and detecting the intent of stripped IoT malware binaries using function signature libraries and 14 defining Linux malware features with corresponding regular expressions. We publish a dataset with 65,956 IoT malware binaries detected over 14 years, containing 1006 unique malware threat labels designed for 15 different architectures. Our results indicate that our feature-specific regular expressions can detect the intent of an IoT malware binary. However, further work on function signature matching is needed to recover a feature-revealing symbol table in stripped IoT malware binaries.**

I. Introduction

Over the last decade, IoT malware has accompanied the rapid growth of embedded devices. Additionally, IoT malware has grown in sophistication to adopt detection prevention techniques and target broader attack vectors [1]. This problem is further problematized by outdated IoT build environments deployed across multiple architectures [2]. The current range of attack methods used by IoT malware includes DDoS, cryptocurrency mining, and phishing techniques [3]. The range of attacks is expected to adopt traditional botnet methods, such as spyware and ransomware [4].

Previous works have mainly focused on the classification and detection of IoT malware through signature matching, honeypots, call graphs, binary similarities, and network-based approaches [1], [3], [5]–[8]. Classification of IoT malware is a popular topic when tracking the evolution of IoT malware. However, classification of IoT malware is a complicated topic. IoT malware changes constantly, either because of an expanded threat landscape, new devices, new firmware versions, unpatched vulnerabilities, or malware authors/researchers publishing malware source code online. Classifying a new malware binary based on heuristic signature matching proves insufficient because novel malware families and variants do not have a signature and tend to be classified based on the closest match.

Malware researchers perform detection of IoT malware when the malware has already been installed on the device, leaving the device vulnerable to attacks such as persistence, evasion, and information gathering [9]. During the window of vulnerability, the malware can infect the configuration files, making reboots useless as the malware still exists on the firmware when rebooted. The malware can also run system scans to determine if the device is running any analysis tools or if the underlying system is an actual device, sandbox, or honeypot. Such a system scan allows the malware to launch attacks based on the observations [9].

Static binary analysis of malware binaries can be sufficient in detecting what malware might attempt to do during the window of vulnerability. However, the security community predominantly relies on manual analysis of IoT malware, which is infeasible on a large scale and burdensome to regularly maintain [1]. Vignau, Khoury, and Hallé [4] proposed an approach to use the characteristic features of 16 well-known IoT malware families to classify malware based on assumed behavior and represent the evolution of IoT malware over time. Their work details specific protocols and exploits detected in the most widespread IoT malware families. Therefore, their work fails to detect novel malware features as they are not yet on their list of features. Cozzi et al. [1] proposed an approach to outline similarities between malware families through binary diffing to track the relationships, evolution, and variants related to IoT malware families over 3.5 years. IoT malware is rapidly developing [7], so their work is likely to be outdated quickly as new communication protocols, architectures, and device characteristics are developed [3]. Additionally, Cozzi et al.'s work relied on VirusTotal as their data source, where the API allowing file downloading is not accessible to the broader public and is, therefore, not shared. Our work takes a different approach from current IoT malware research, as we construct a diverse, open-source dataset with unique IoT malware binaries over 14 years. The dataset consists of executable files for 1006 unique malware threat labels designed for 15 different architectures, making it longitudinal and robust. We construct a set of 14 defining Linux malware features with a corresponding regular expression and evaluate how well our regular expressions are at detecting features in the stripped binaries in our dataset. This paper makes the following contributions:

1) We construct and publish a longitudinal, open-source, labeled dataset of IoT malware, consisting of 65,956 executable files detected over 14 years. The dataset is available at *https://research.fit.edu/iot*.
2) We design a robust dataset of function signature libraries for seven different CPU architectures across ten distinguished malware families and 14 defining Linux

features.

3) We demonstrate the depth of the dataset by analyzing 700 stripped IoT malware binaries, recovering the symbol tables, and detecting the intent for 111 unique malware threat labels from our dataset.

II. Motivation & Background

A. Motivation

IoT malware is an important and rapidly developing phenomenon because embedded devices are resource-constrained [8] and might be unable to run anti-virus malware solutions comparable to the ones used to protect PCs and desktop systems [1]. The rapid development results from embedded devices being popular targets and malware researchers and authors publishing source code for feature-rich IoT malware families online [4]. Novel IoT malware families and variants, excluding IoT malware explicitly designed to exploit 0-day vulnerabilities, therefore adopt many of the features seen in known IoT malware families [8], such as usage exploitation, file downloading and replacement strategies, permission changes, and execution process. Analysts currently rely on manual analysis of binaries [1], so these features can be detected by reverse-engineering the binary when it, in the best case, is dynamically linked and not stripped. However, this is rarely the case. IoT malware is usually designed to be run on multiple architectures and systems and is, therefore, statically linked, resulting in a binary with thousands of functions. Also, malware authors tend to strip the symbol table from the binaries as the program is developed with malicious intent, and they wish to hide their purpose. Stripped binaries are an issue because reverse engineering tools cannot put a name to the functions in the binary. If we combine this with static linking, we have thousands of functions without a name. Putting a name to functions and detecting the features in these binaries is essential to accurately detect the threat landscape and figure out how malware might attempt to exploit a system. To address this challenge, we investigate if we can recover the symbol table and detect the intent in stripped IoT malware binaries using function signature matching and 14 defining Linux malware features.

B. Architectures

When the rapid growth of embedded devices started, the malware landscape expanded to include IoT devices. Malware developers began to design malware that could run on MIPS and MIPSEL devices, not just the x86-flavored architectures broadly used in personal computers [7], [9]. With new architectures, novel malware families attempted to download and execute every variant of the malware on the device until the correct architecture-specific variant got accepted [4]. Later, Rematien and Mirai's malware families introduced a scanning feature where they scanned the target device and only downloaded the correct malware on the device. In 2012, the botnets Carna and Aidra expanded the use of IoT malware to include all other commonly used architectures [4]. The botnets introduced cross-compilation capabilities, which enabled running the malware on different architectures, such as x86_64 or ARM, without creating architecture-specific designed binaries.

TABLE I
The binaries in our dataset are designed for many architectures, with ARM, Intel, and MIPS as the dominating ones.

Architecture	Count
ARM	18,842 (28.57%)
Intel 80386	13,875 (21.04%)
MIPS R3000	11,183 (16.96%)
Renesas SH	5,295 (8.03%)
Motorola 68000	5,173 (7.84%)
PowerPC or cisco 4500	4,522 (6.86%)
AMD x86_64	3,487 (5.28%)
Sparc	3,302 (5.01%)
AArch64	92 (0.14%)
ARC Cores Tangent-A5	90 (0.14%)

Table I shows ten of the architectures in our dataset. We retrieve the architecture for each binary in the dataset from VirusTotal [10]. The dataset contains only ELF binaries, but ELF binaries do not only target embedded devices. ELF binaries are also utilized for Linux desktop systems and other UNIX-like systems. The different target systems for ELF files mean that some of the binaries in our dataset are likely designed for Linux desktop systems, not embedded devices.

Primarily Intel and AMD architectures are widely used in Linux desktop systems. It is challenging to determine if a binary compiled for Intel and AMD architectures is designed for Linux desktop systems or embedded devices [1]. Given the difficulty, some academic papers on IoT malware exclude these architectures from the analysis [1]. Other academic papers purposely only focus on MIPS, ARM, and x84_64 architectures because they are the most popular for embedded devices [5].

There are also arguments for why analysts should include ELF binaries designed for Intel and AMD architectures. IoT malware tends to adopt features seen in malware targeting personal computers. Two examples of these features are virtual evasion and cryptocurrency mining. The malware family Darlloz contained the cryptocurrency mining feature when it was introduced in 2014, which allowed it to mine cryptocurrencies with the computing power of the infected device [4]. The Tsunami variant Amnesia contained the virtual evasion feature to delete itself and the entire file system from the infected device if the target device ran a VM [4], [11]. Therefore, including binaries designed for Intel and AMD architectures is interesting when analyzing features, as features in binaries developed for Linux desktop malware are likely to be adopted by malware families targeting embedded devices.

C. Binary Analysis and Similarities

Analysis of binary files is an effective technique when studying malware. Binary analysis allows malware researchers

TABLE II
THE COMPILATION OF THE BINARIES IN OUR DATASET SHOWS THAT THE NUMBER OF STRIPPED AND NON-STRIPPED BINARIES IS VERY SIMILAR.

Symbol Table	Count
Not stripped	44.17%
Stripped	44.04%
No section header	10.61%
Missing section header	1.00%
Corrupted section header	0.16%

TABLE III
THE BINARIES IN OUR DATASET ARE MOSTLY STATICALLY LINKED, ALLOWING CROSS-COMPILATION AND EXECUTION ON SEVERAL ARCHITECTURES.

Linking	Count
Statically linked	58,994 (89.44%)
Dynamically Linked	6,728 (10.20%)
Not specified	234 (0.35%)

to study the control flow of executable files and get an overview of the threat landscape posed by malware. Vendors responsible for devices and services can use the information obtained from a binary analysis to patch known or unknown vulnerabilities exploited by malware authors.

One of the goals of binary analysis is to generate human-readable code at the assembly level from executable code [12]. Generating human-readable code is particularly interesting when working on malware because the malware source code is not always made available to the public. Vignau, Khoury, and Hallé [4] found that five of the 16 most popular IoT malware families had publicly available source code in 2019. Therefore, using binary analysis to analyze the malware threat landscape is an effective technique that does not rely on malware authors and researchers publishing the source code.

Binary Diffing: Binary diffing is a binary analysis technique used to detect similarities between binaries at the assembly level [1]. Detecting similarities between binaries at the assembly level removes the issue of equivalent functions with misleading/different names and stripped binaries. Table II shows that 44.17% of the binaries in our dataset are non-stripped. We can use binary diffing to detect similarities between non-stripped and stripped binaries to detect interesting information about stripped binaries that we would not easily detect through manual binary analysis. An issue with binary diffing on IoT malware is that the binaries are often statically linked. Static linking means all the code, including library code, is included in one executable file [13]. As shown in Table III, 89.44% of the binaries in our dataset are statically linked. Performing binary diffing on statically linked binaries is complicated since two statically linked binaries can falsely be deemed similar because they contain the same libraries, such as libc libraries [1].

NinjaDiff [14] is a plugin created to do binary diffing in Binary Ninja. The plugin constructs a directed graph mirroring the CFG of each function to detect resemblances between two binaries [15]. This approach removed issues related to specific compilers reordering functions within a binary and heuristic hash diffing [15]. However, Riverloop Security developed Ninjadiff to support the x86/x86_64 and ARM architectures, which excludes many binaries from our dataset. Performing automated binary diffing on a set of binaries is also very time-consuming. Each file has to undergo the analysis process to create complete control flow graphs and function detection before going through the diffing process. Time inefficiencies can be defended because automated binary diffing on non-stripped versus stripped binaries can give us an overview of the evolution of malware families and exactly where novel malware families borrow code from.

Function Signature Matching: The increased amount of publicly available IoT malware source codes has resulted in extensive code reuse and code borrowing [1]. Code reuse and borrowing between malware families can be used as an advantage for IoT malware researchers. We can create function signatures for malware binaries and use constructed function signature libraries to put a name to functions in stripped binaries. Vector35 implemented a signature kit plugin [16] for Binary Ninja, where the goal was to cope with the issues related to analyzing statically linked and stripped binaries [17]. One of the issues is differentiating between external library functions and user-defined functions with a statically linked and stripped binary [17]. Identifying user-defined functions is particularly interesting when analyzing code reuse and the evolution of features used in IoT malware. A signature matcher identifies library code well because the user does not alter it. Identifying user-defined functions is more difficult because reused and borrowed code can easily be modified. The signature kit plugin developed by Vector35 requires that the function in the stripped binary matches all the call-sites and callees of the signature for it to be matched [17]. Malware authors can therefore avoid function signature detection by adding function calls to empty functions, as it would not match the FCG of the signature.

III. KEY IDEAS

Constructing a Dataset of IoT Malware: Accessing IoT malware is challenging due to constraints added by the major malware providers, such as VirusTotal. Downloading malware samples from VirusTotal is possible only with certain subscriptions and, therefore, unavailable to most people. Other malware providers either rate limit the number of files one can search for [10], [18] or have samples of IoT binaries mixed with binaries targeting all kinds of systems [19]. Restrictions added by malware dataset distributors make it much more difficult for malware researchers to access a dataset with IoT malware. Building a structured open-source dataset of IoT malware binaries can promote further research and development on IoT, including dynamic malware analysis,

TABLE IV
BINARIES CLASSIFIED AS MIRAI OR GAFGYT MAKE UP APPROXIMATELY 86% OF OUR DATASET.

Threat Label	Count
Mirai	35,955 (54.51%)
Gafgyt	21,692 (32.88%)
Tsunami	1,238 (1.88%)
Benign	1,215 (1.85%)
Generica	934 (1.43%)
Rootkit	319 (0.48%)
Dofloo	318 (0.48%)
Flooder	161 (0.25%)
Dnsamp	157 (0.23%)
Jiagu	141 (0.21%)
Total	65,956

security hardening, threat modeling, vulnerability assessments, and malware detection and prevention.

One of the key contributions of our work is that we construct a robust and diverse open-source dataset of IoT malware. The malware binaries are primarily collected from MalwareBazaar [18], VirusShare [19], VxUnderground [20], and Contagio [21]. After constructing the dataset, we use VirusTotal's [10] universal 'Search & Metadata' API endpoint to retrieve metadata and store it in a database. As seen in Table IV, the dataset contains 65,956 unique binaries. We use the MD5 hash for the binary as a primary key to ensure all the inputs to the dataset are unique. We also store metadata such as architecture, entry point, first submission date, last analysis date, number of malicious votes on VirusTotal, threat label, compilation, file size, and linking. The dataset is not the largest IoT malware collection addressed in related research. However, it differs from the publicly available malware collections by being structured on architecture and threat label, longitudinal, and IoT-focused. Figure 1 shows the distribution of when the binaries in our dataset were submitted to VirusTotal.

Function Signature Matching for IoT Malware: Reverse engineering and analyzing a stripped binary with statically linked libraries is a problematic task [17]. IoT malware binaries are usually statically linked so they can run on various system configurations [1], and stripping a binary is a way for malware authors to make static analysis of their files much more difficult. In our dataset, 24,896 files (37.75%) are statically linked and stripped. To simplify the analysis of statically linked and stripped binaries, we apply function signature matching to the analysis process to see if we can put a name to the functions whose name has been stripped. We use the dataset we have constructed and create function signatures for the most characteristic malware families in our dataset. When running the signature matcher in Binary Ninja, it will match against signatures related to the platform the binary is designed for [17].

Function signatures have the same limitations as YARA rules. A function signature cannot recognize a function it has never seen before, so modified functions will go undetected. Maintaining the signature library to account for new compiler versions and architectures is also necessary. However, given the extensive amount of code reuse between the different IoT malware families [1] and the number of malware variants that rose from publicly available source codes [4], we can assume that function signatures for IoT malware families can match user-defined code as well as library code in stripped binaries. Therefore, we create signatures for IoT malware families and observe how many functions we can recognize in a set of stripped binaries.

Feature-Based Static Analysis of Stripped IoT Malware: One of the limitations of using signatures, both for malware classification and functions, is that signatures cannot match samples it has not seen before. If a malware author modifies a function with a corresponding signature, the signature might not be able to match the function. This matching approach is problematic because malware authors use obfuscation techniques to hide what they are doing. So, even with code reuse, the signature matching results might be limited for certain files. The signature matching results are also likely to be limited for novel IoT malware families exploiting 0-day vulnerabilities as there tend to be device-specific targets and less code reuse. Fortunately, strings remain in the binary when a binary has been stripped. So, even if the results from the signature matcher are limited, we can still use strings to predict what the malicious executable might try to do once a target has been compromised. We, therefore, put together a set of features with corresponding regular expressions to detect features in binaries.

Some works have focused on the design, implementation, and characteristics of Linux malware [9], [22]. We assemble a set of general Linux malware feature categories discussed in these works and construct a regular expression corresponding to each category. These categories include subsystem initialization, time-based execution, user file alternation, file infection and replacement, sandbox detection, process renaming and termination, privilege escalation, anti-debugging, process injection, cryptography, stalling code, configuration files, network information, and flooding [4], [9], [22]. Additionally, we store strings related to IPv4 addresses in 255.255.255.255 notation [23], IPv6 addresses in standard notation [23], requests, and infection method. If the signature matcher can match functions in the binary, it will add the matched functions to the symbol table. We can then loop through the functions in the symbol table and run the recovered function names against the regular expressions. The results from such a feature-based static analysis can be utilized by IoT professionals to perform vulnerability assessments and stay up-to-date with the quickly evolving threat landscape.

IV. EXPERIMENTS

Experiment 1: Function Signature Generation and Matching: We generate signatures for a large set of non-stripped files in our dataset. To generate signatures, we make additions to

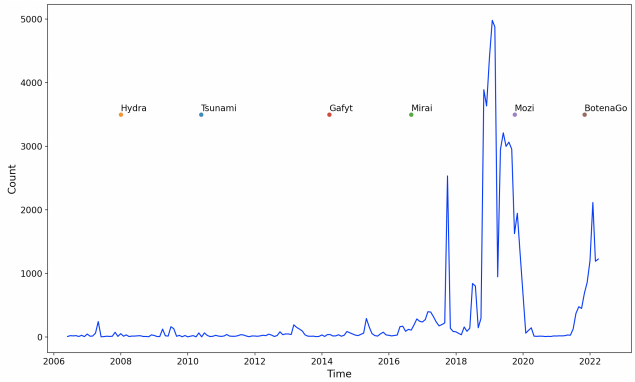

Fig. 1. Unique files submitted to VirusTotal grouped by month combined with the rise of dominant malware families. The months after the release of Mirai's source code saw a spike in novel malware variants. The end of 2018 and the beginning of 2019 saw the most significant spike in novel malware variants to date.

the signature kit plugin, sigkit, created by Vector35 [16]. Our ultimate goal for this experiment would be to recognize user-defined code in stripped binaries, so we group the signatures based on the threat label in our dataset. Generating signatures is a time-consuming process, so we create signatures for non-stripped binaries between 300 kB and 2 MB in size. The sigkit plugin makes it easy to combine signature libraries, so we do not have to worry about whether the binary is dynamically or statically linked. A signature for a dynamically linked binary can be combined with a signature for a statically linked binary, resulting in a signature that can match both library code and user-defined code.

Table V shows the signatures created for the different platforms. We also observe that Binary Ninja differentiates between 'linux-armv7' and 'armv7' platforms. The difference between 'armv7' and 'linux-armv7' is that Binary Ninja could not recognize the target OS by looking at the standard system calls in binaries with 'armv7' platform [24]. The malware labels were selected based on malware families with the most amount of features [4], representation in our dataset, and average binary file size. Additionally, we create a combined function signature library for malware families less represented in the dataset.

Experiment 2: Feature Analysis: We run the signature matching algorithm and apply feature detection based on strings found in the binary and functions added to the symbol table by the signature matching algorithm. Determining the success of this experiment is difficult if we only look at the overall number of features we detect. Therefore, we group the stripped binaries by year and look at the evolution and popularity of features over the last ten years. For consistency, we analyze files between 300 kB and 2 MB in size as we did in the previous experiment. To detect features, we perform a search, and if there is a match, we store the entire string/function containing the matched substring. Storing the entire string allows us to manually see how the malware intends to utilize a feature, such as how malware will utilize an IP address or the name of the file a malware is trying to remove, modify or replace. We also look at features detected in malware families representing different malware categories.

As mentioned earlier, we store IPv4 addresses in 255.255.255.255 notation [23], IPv6 addresses in standard notation [23], requests, and infection method. However, our evolution evaluation does not include IP detections, requests, and infection methods. These detections will rather be used to evaluate the adoption of IPv6 and reveal which services malware authors try to access. For our evolution evaluation, we use the general Linux malware categories discussed in previous works [4], [9], [22].

V. Findings

Finding 1: Function Signature Matching: Table VI shows the signature matching results for statically linked versus dynamically linked files in our dataset. As expected, the signature matching algorithm performed better at recognizing LibraryFunctionSymbols. The majority of the recognized functions being LibraryFunctionSymbols is likely because the algorithm loads all signatures for a specific platform when matching and will therefore match library code without regard to the threat label [17]. Additionally, user-defined functions are modified more frequently compared to library functions. However, given the vast amount of code reuse [1] and the fact that we generated signatures mainly by threat label instead of a random selection, as seen in Table V, we expected a more significant portion of FunctionSymbols in the results.

Table VII shows the signature matching results grouped by threat label and ordered by the total number of matched functions. Interestingly, only four out of ten threat labels in Table VII are also in Table V. We can also observe that these four threat labels, particularly Mirai and Dnsamp, gave limited results. The signature matcher matched 0 FunctionSymbols for the Dnsamp family despite possessing a 768 kB extensive signature library for the particular malware family. The absence of FunctionSymbols for Dnsamp can support the claim that malware labels tend to be coarse-grained [1], as the files in the signature library or the matching process might be incorrectly classified. It can also support the claim that publicly available source codes have resulted in the rapid development of novel IoT malware with roots in released malware source codes [4], resulting in modified functions remaining undetected in a signature matching algorithm. The results prove the difficulties regarding matching user-defined code in stripped IoT binaries. Relying solely on signature matching to analyze stripped binaries, even when targeting user-defined functions, proves insufficient to reveal the intention of an IoT binary.

Finding 2: IoT Malware Feature Correlations: Figure 2 shows the results from analyzing the features found in stripped IoT malware binaries between 2012 and 2022. An interesting observation in this chart is the correlation between process injection, cryptography, network information, and configuration files. The similarities between Table VIII and Figure 2

TABLE V

WE GENERATED MALWARE SIGNATURES FOR SEVEN DIFFERENT PLATFORMS IN OUR DATASET. THE PLATFORM FOR A BINARY IS DETERMINED BY LOOKING AT THE SYSTEM CALLS IN THE BINARY.

Malware	linux-mips	linux-mipsel	linux-x86	linux-x86_64	linux-armv7	armv7	linux-armv7eb
Dofloo		✓	✓		✓	✓	
Benign		✓	✓	✓	✓		
Rootkit			✓	✓	✓	✓	
Dnsamp	✓	✓	✓		✓	✓	
Jiagu			✓		✓	✓	
Generica	✓	✓	✓		✓	✓	✓
Tsunami	✓	✓		✓	✓	✓	
Gafgyt	✓	✓	✓	✓	✓	✓	
Mirai	✓	✓	✓	✓	✓	✓	
VPNFilter		✓					
Other	✓	✓	✓	✓	✓	✓	

TABLE VI

THE SIGNATURE MATCHING RESULTS FOR STATICALLY LINKED VERSUS DYNAMICALLY LINKED FILES IN OUR DATASET SHOW A MORE SIGNIFICANT NUMBER OF MATCHED FUNCTIONS IN STATICALLY LINKED BINARIES THAN DYNAMICALLY LINKED ONES.

Linking	FunctionSymbols	LibraryFunctionSymbols	ImportedFunctionSymbols	Total Matched	Total Functions	Files
Statically Linked	20,402	172,350	0	192,752 (12.6%)	1,570,900	481
Dynamically Linked	5,030	7,024	12	13,672 (2.2%)	644,408	219
Total	25,432	180,980	12	206,424 (9.5%)	2,215,342	700

TABLE VII

SIGNATURE MATCHING RESULTS GROUPED BY THREAT LABEL AND ORDERED BY THE TOTAL NUMBER OF MATCHED FUNCTIONS SHOW HIGHER RESULTS FOR MALWARE FAMILIES UTILIZING SCANNING AND DDOS TECHNIQUES.

Threat Label	FunctionSymbols	LibraryFunctionSymbols	ImportedFunctionSymbols	Total Matched	Total Functions
Mirai	4,359	10,350	0	14,709 (7.0%)	211,661
SSHscan	1,160	10,418	0	11,578 (22.7%)	51,339
Generica	1,618	8,152	0	9,770 (11.2%)	86,944
DDoStf	69	8,479	0	8,479 (40.6%)	20,854
XORDDoS	511	5,702	0	6,213 (23.4%)	26,536
Dnsamp	0	6,265	0	6,265 (4.6%)	136,624
Portscan	305	2,818	0	3,123 (20.1%)	15,519
Gafgyt	313	2,706	0	3,019 (20.4%)	14,814
Lupper	156	1,771	0	1,927 (21.9%)	8,797
Mare	69	1,155	0	1,224 (25.6%)	4,787

show that two of the three cryptocurrency mining families have all four correlated features. The correlation observed between Table VIII and Figure 2 is particularly interesting because Darlloz was the first cryptocurrency mining IoT malware. The malware family was introduced in 2014, just months before the spike for these particular features. We can assume that cryptocurrency mining IoT malware binaries will likely contain these features with these observations.

We also observe a correlation between time-based execution and flooding. This correlation is interesting because devices in a DDoS botnet tend to flood services simultaneously. Another interesting observation is that 11/14 features had an upwards trend after 2016, which supports the claim that the Mirai botnet is one of the botnets with the most amount of features [4] and that the publication of the source code resulted in a large number of variants. [25].

Researchers and professionals can correlate features detected in IoT malware binaries to understand how targeted devices are exploited and prioritize actions to mitigate a potential attack. Table VIII looks at features detected in malware families representing different malware categories. These malware categories are DDoS, ransomware, and cryptocurrency mining. An unexpected observation is that we could not detect features related to user file alternation for the ChaCha family, as ransomware is known to encrypt files on the target device. Similarly, we expected to detect flooding characteristics for the XORDDoS family, as DDoS attacks are known to reduce the availability of services by flooding the target. Across all three malware categories, we could detect process remaining and termination, a popular obfuscation feature attempting to spoof a process name to make it look harmless [4]. Despite not detecting all of the expected features, the correlations in Figure 2 compared to observations in Table VIII illustrate which features binaries from different malware categories are likely to contain and, therefore, prove sufficient to detect the intent of an IoT malware binary.

Finding 3: IPv4 and IPv6 Detection: As previously discussed, many IoT malware families rely on scanning the

TABLE VIII
FEATURE DETECTION RESULTS FOR MALWARE FAMILIES FROM DIFFERENT IOT MALWARE CATEGORIES SHOW CONSISTENCIES FOR MALWARE FAMILIES FROM THE SAME MALWARE CATEGORIES.

	DDoS			**Ransomware**	**Cryptocurrency Mining**		
	Dnsamp	Mirai	XORDDoS	Chacha	Cryptonight	Risk	XMRig
Subsystems Initialization		✓	✓	✓			
Time-based Execution	✓	✓	✓	✓			
User File Alternation	✓	✓	✓				
File Infection and Replacement	✓	✓	✓		✓	✓	✓
Sandbox Detection							
Process Renaming and Termination	✓	✓	✓	✓	✓	✓	✓
Privilege Escalation		✓	✓			✓	✓
Anti-Debugging							
Process Injection	✓	✓	✓	✓		✓	✓
Cryptography	✓	✓	✓	✓	✓	✓	✓
Stalling Code			✓	✓	✓	✓	✓
Configuration Files	✓	✓	✓	✓		✓	✓
Network Information	✓	✓	✓	✓	✓	✓	✓
Flooding	✓	✓			✓	✓	✓

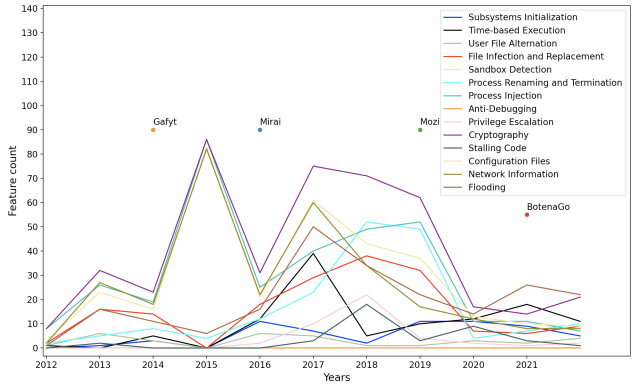

Fig. 2. Feature analysis illustrating the number of features detected each year between 2012 and 2022 shows an increase in cryptography, network information, process injection, and configuration files after the rise of Gafgyt. The overall feature landscape increased after the rise of Mirai.

IP space for vulnerable devices. IPv6 makes scanning this IP space much more difficult [26] because of the much larger address space. An interesting observation from our IPv4 and IPv6 detections is that we detected 1,594 IPv4 and 0 IPv6 addresses. The absence of IPv6 detections can indicate the complexity of scanning IPv6 addresses and that current scanning techniques are not efficient enough to be included in current IoT malware. Nonetheless, many features seen in IoT malware have become increasingly complex, including known IP space scanning algorithms. Therefore, we may see malware authors introducing efficient IP space scanning algorithms, including IPv6 scanning techniques, in future novel IoT malware even though we could not detect it in this experiment.

VI. LIMITATIONS

Accuracy of Feature Detection on Symbols: Our feature detection approach relies on running strings and function names in a binary on a set of regular expressions to determine a feature. However, these strings and function names are not always straightforward. A manual analysis of a set of binaries shows that some malware authors tend to change a few characters in a function name. We assume the modifications are made to limit automatic binary analysis classifications while keeping the intent clear for prospective malware authors. Some examples of these obfuscation techniques are different versions of functions with flooding and spoofing purposes. An observation is that these functions are occasionally named FL00D, Flkkd, and SP00F. We wanted to match these function names, so we had to generalize the regular expressions.

Similarly, we wanted to detect IPv4 and standard IPv6 addresses in a binary. IP detection requires general regular expressions, resulting in false positives such as software version numbers matching the same pattern. Given the generalization of regular expressions and obfuscation techniques, our feature detection approach will result in some false positives and false negatives.

Like previous works, our feature detection approach cannot detect novel IoT malware features even though we take a more general approach to cover broader Linux malware features. Novel malware features must be appended to our set of features as IoT malware adopt new features and become more complex.

Determining if a Malware Family is Designed for IoT Devices: Determining if a malware family is originally designed for IoT devices or Linux desktop systems is difficult as the threat label does not have the target system included in its metadata. From Table VIII, we see that we analyzed at least one binary from the ChaCha malware family. The ChaCha malware family is ransomware known to target Windows desktop systems [27]. Therefore, we can assume binaries representing the Chacha family in our dataset target Linux desktop systems. From observations in our dataset, this specific malware family also targets MIPS R3000, ARM, Intel 80386, and AMD x86_64. There is a chance the malware family would have been included in the analysis even if we had decided to exclude Intel and AMD architectures.

Creating a dataset for threat labels with a corresponding target system is one possible solution to this limitation. However, it could come short if malware authors refactor the source code for a known malware family targeting Linux desktop systems to target embedded devices. Further work is needed to distinguish binaries designed for embedded devices from those designed for Linux desktop systems, regardless of the architecture.

VII. Future Work

CVE Search Based on Feature Detection Results: Malware families using 0-day exploits to infect a device, such as Gafgyt and Hajime, tend to be more effective and can potentially infect more devices than other malware families as these malware families exploit unknown vulnerabilities in devices or protocols [3], [4]. Designing botnets that leverage 0-day vulnerabilities is usually more time-consuming to develop as one would first need to find a vulnerability to exploit. These attacks are effective until the vulnerability has been detected and patched by the ones responsible for the service. However, these vulnerabilities might not represent an isolated occurrence. They can indicate a more significant systemic design flaw in many IoT devices [28]. Other devices might use the same vulnerable protocols or services. If the service/protocol providers patch an exploitable vulnerability, but the firmware developers do not update the version, the firmware will still be vulnerable to the now-known exploit. Even if firmware developers want to update to a new and more secure version, outdated hardware might prevent such an update [8]. Known vulnerabilities and exploits are usually reported to a CVE system, such as the CVE database hosted by Mitre [29]. Some N-day exploits can also be found ready-to-use in penetration testing tools such as Metasploit [30].

Over the last few years, there has been an increase in botnets relying on CVEs instead of the standard dictionary attack to infect devices [4]. Performing a CVE search based on the symbols and features found in our feature detection experiment could give us an overview of vulnerable devices and the severity of the CVEs exploited in the binary.

Function Signature Matching to Classify User-Defined Functions: As one of the goals of function signature matching algorithms is to avoid false positives [17], we expected that we would not be able to match a majority of the user-defined functions. Therefore further work on the topic is needed to make it easier for reverse engineers to understand the purpose of the functions in a stripped binary. One memory-inefficient approach mentioned by the team at Vector35 is to scrape all the library functions from the apt repositories [17]. One possible addition would be adding confidence levels to the function signature matching algorithm. The algorithm would provide a list of functions with confidence levels above a set threshold without adding these functions to the symbol table. Confidence levels would avoid false positives in the symbol table and could reveal features that would be difficult to detect during a manual analysis of stripped binaries. Advancing in this area could force malware authors to modify borrowed code and push for more efforts to hide the intention of the functions.

Regular Expressions Matching Device Specific Names: An in-depth analysis of the Mirai botnet found that the manufacturers responsible for the most infected devices were Dahua, Huawei, ZTE, Cisco, ZyXEL, and MikroTik [25]. While we manually analyzed a few other IoT malware binaries to understand any obfuscation techniques, we detected that BotenaGo alone targeted 26 different devices from 16 vendors. These vendors were Fiberhome, Vigor, Comtrend, Gpon Fiber, Broadcom, Hongdian, Realtek, Tenda, Totolink, ZyXEL, Alcatel, Lilin Dvr, Zte, Linksys, Netgear, and Dlink. A similar analysis of a Mirai binary in our dataset found 28 vendors. These were Alcatel, Asus, Barracuda, Beckhoff, Belkin, Beward, Citrix, Cloud, CTEK, DD-WRT, Dell, Dlink, Fritz, Geutebruck, Homematic, HooToo, Linksys, Mitel, Nagios, Oracle, QuickTime, Realtek, Sapido, Seowon Intech, VMware, Xfinity, Yealink, and ZeroShell. Building a dataset of IoT devices and vendors can therefore be used to detect which devices a binary might try to target and help businesses perform security hardening and threat modeling on their devices based on the attack paths exploited by the malware. However, such an analysis will only be feasible with non-stripped binaries, as the vendor name is usually in the function name.

VIII. Related Work

Cozzi, Graziano, Fratantonio, and Balzarotti [9] is the closest to our work as our feature detection approach builds upon the work they did. They went in-depth on features used by Linux malware and corresponding characteristics we could use in our regular expressions. While they went deep into all possibilities for Linux malware, such as anomalous and invalid ELF files, packing, and standard libraries, we focused on the common techniques related to evasion, deception, information gathering, process interaction, privilege, and persistence.

Vignau, Khoury, and Hallé [4] proposed an approach to use the characteristic features of 16 well-known IoT malware families to classify malware based on assumed behavior and represented the evolution of IoT malware over time. Different from our work, they look at how IoT malware families have influenced successive ones. For this process, they gathered information on the different malware families from different academic research papers and press articles. To determine the set of features for a particular malware family, they rely on at least two academic papers or technical reports. This work is promising because it looks at how malware families borrow features from each other. Some of the features they look into overlap with the ones we used in our work, such as flooding attacks, network information, privilege escalation, and process altering. Unlike our work, they detail the specific protocols and exploits seen in the most widespread IoT malware families. Our set of features is based on general Linux malware features to recognize defining malware features and detect features IoT malware might adopt from PC malware.

Cozzi et al. [1] proposed an approach to detect similarities between malware families through binary diffing to track the relationships, evolution, and variants related to IoT malware families. While we do not look at the evolution of malware families, we look at the evolution of features used by IoT malware and the correlation between the features. Similar to our work, they collect a dataset with 93,652 binaries over 3.5 years. Their dataset is significantly larger than our dataset. However, while we collected data from public data sources, their source requires a VirusTotal subscription that the broader public cannot obtain. Our work is a step toward constructing a publicly available IoT malware dataset for future researchers.

IX. Conclusion

The last few years have seen an increase in works on static analysis of IoT malware [1], [4], [7], [9], [22]. However, these works rely on subscription-protected data sources, and the approaches tend to be outdated quickly. Our work provides a longitudinal, open-source, labeled dataset of IoT malware features. We construct a dataset of 65,956 executable files and 14 defining Linux malware features with corresponding regular expressions. To demonstrate the depth of the dataset, we recovered partial symbol tables and intent for 700 stripped IoT malware binaries. Our results indicate that our feature-specific regular expressions can detect the intent of an IoT malware binary. However, further work on function signature matching is needed to recover a feature-revealing symbol table in stripped IoT malware binaries.

Acknowledgements

This material is based upon work supported in whole or in part with funding from the Office of Naval Research (ONR) contract #N00014-21-1-2732. Any opinions, findings, conclusions, or recom- mendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ONR and/or any agency or entity of the United States Government.

References

[1] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bilge, and D. Balzarotti, "The tangled genealogy of iot malware," in *ACSAC '20*. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–16. [Online]. Available: https://doi.org/10.1145/3427228.3427256

[2] D. Andriesse, A. Slowinska, and H. Bos, "Compiler-agnostic function detection in binaries," in *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, 2017, pp. 177–189.

[3] Y. Xu, Y. Jiang, L. Yu, and J. Li, "Brief industry paper: Catching iot malware in the wild using honeyiot," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 433–436.

[4] B. Vignau, R. Khoury, and S. Hallé, "10 years of iot malware: A feature-based taxonomy," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2019, pp. 458–465.

[5] C. Tamás., D. Papp., and L. Buttyán., "Simbiota: Similarity-based malware detection on iot devices," in *Proceedings of the 6th International Conference on Internet of Things, Big Data and Security - IoTBDS,*, INSTICC. SciTePress, 2021, pp. 58–69.

[6] R. Searles, L. Xu, W. Killian, T. Vanderbruggen, T. Forren, J. Howe, Z. Pearson, C. Shannon, J. Simmons, and J. Cavazos, "Parallelization of machine learning applied to call graphs of binaries for malware detection," in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2017, pp. 69–77.

[7] Q.-D. Ngo, H.-T. Nguyen, H.-L. Pham, H. H.-N. Ngo, D.-H. Nguyen, C.-M. Dinh, and X.-H. Vu, "A graph-based approach for iot botnet detection using reinforcement learning," in *International Conference on Computational Collective Intelligence*. Springer, 2020, pp. 465–478.

[8] F. Ding, H. Li, F. Luo, H. Hu, L. Cheng, H. Xiao, and R. Ge, "Deeppower: Non-intrusive and deep learning-based detection of iot malware using power side channels," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 33–46. [Online]. Available: https://doi.org/10.1145/3320269.3384727

[9] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 161–175.

[10] VirusTotal, "Search & metadata," Available from VirusTotal universal Search & Metadata API endpoint. [Online]. Available: www.VirusTotal.com

[11] C. Xiao and C. Zheng, "New iot/linux malware targets dvrs, forms botnet," Available from Palo Alto Network's Unit 42 research posts, Apr 2017. [Online]. Available: https://unit42.paloaltonetworks.com/unit42-new-iotlinux-malware-targets-dvrs-forms-botnet/

[12] Hexrays, "Ida pro," Available from Hexrays's products selection, 2022. [Online]. Available: https://hex-rays.com/ida-pro/

[13] IBM, "When to use dynamic linking and static linking," Available from IBM AIX 7.2, October 2021. [Online]. Available: https://www.ibm.com/docs/en/aix/7.2?topic=techniques-when-use-dynamic-linking-static-linking

[14] R. Security, "Ninjadiff," Available from riverloopsec/ninjadiff GitHub repository. [Online]. Available: https://github.com/riverloopsec/ninjadiff

[15] R. O'CONNELL, "Ninjadiff - open source binary hashing," Available from Riverloop' blog post. [Online]. Available: https://www.riverloopsecurity.com/blog/2021/02/binary-diffing/

[16] Vector35, "Vector35/sigkit: Function signature matching and signature generation plugin for binary ninja," Available from Vector35/sigkit GitHub repository. [Online]. Available: https://github.com/Vector35/sigkit

[17] S. Tong, "Binary ninja - signature libraries," Available from Binary Ninja blog posts, Mar 2020. [Online]. Available: https://binary.ninja/2020/03/11/signature-libraries.html

[18] Abuse, "Malwarebazaar," Available from MalwareBazaar database search. [Online]. Available: https://bazaar.abuse.ch/api/

[19] VirusShare, "Virusshare.com - because sharing is caring," Available from VirusShare torrents. [Online]. Available: http://www.VirusShare.com./

[20] Vx-underground, "Vx-underground - malware samples," Available from Vx-Underground malware samples. [Online]. Available: https://samples.vx-underground.org/samples/Families/

[21] Mila, "Amnesia / radiation linux botnet targeting remote code execution in cctv dvr samples," Available from Contagio's malware dump, Oct 2019. [Online]. Available: http://contagiodump.blogspot.com/2019/10/amnesia-radiation-linux-botnet.html

[22] G. Kambourakis, M. Anagnostopoulos, W. Meng, and P. Zhou, *Botnets: Architectures, countermeasures, and challenges*. CRC Press, 2020.

[23] J. Goyvaerts and S. Levithan, *Regular expressions cookbook*. O'Reilly, 2012.

[24] J. Wiens, "Binary ninja - discussion 3039," Available from Binary Ninja API discussion 3039. [Online]. Available: https://github.com/Vector35/binaryninja-api/discussions/3039

[25] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[26] R. Perdisci, T. Papastergiou, O. Alrawi, and M. Antonakakis, "Iotfinder: Efficient large-scale identification of iot devices via passive dns traffic analysis," in *2020 IEEE European Symposium on Security and Privacy (EuroS P)*, 2020, pp. 474–489.

[27] A. Mundo, "Ransomware maze," Available from McAfee Labs blog posts, Mar 2020. [Online]. Available: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/ransomware-maze/

[28] T. OConnor, W. Enck, and B. Reaves, "Blinded and confused: Uncovering systemic flaws in device telemetry for smart-home internet of

things," in *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. Miami,FL: ACM, May 2019.

[29] MITRE, "Cve - mitre," Available from MITRE CVE Search List, Jan. 2022. [Online]. Available: https://cve.mitre.org/cve/search_cve_list.html

[30] Rapid7, "Metasploit — penetration testing software," Available from Metasploit home page. [Online]. Available: https://www.metasploit.com/