# An Evolutionary Study of IoT Malware

Huanran Wang, Weizhe Zhang, *Senior Member, IEEE*, Hui He, *Member, IEEE*, Peng Liu, *Member, IEEE*, Daniel Xiapu Luo, Yang Liu, Jiawei Jiang, Yan Li, Xing Zhang, Wenmao Liu, Runzi Zhang, *Member, IEEE*, and Xing Lan

***Abstract*—Recent years have witnessed lots of attacks targeted at the widespread Internet of Things (IoT) devices and malicious activities conducted by compromised IoT devices. After some notorious IoT malware released their source code, many new variants emerge, which are usually more powerful and stealthy. Although numerous existing studies have analyzed some exposed families, there is a lack of systematic study to make full use of them, which can be a fundamental step for provenance, triage, labeling, lineage analysis, and authorship attribution. The key challenge of conducting an IoT malware evolutionary study is how to collect sufficient and accurate information about malware and identify the relationships among them. In this article, we take the first step to investigate the IoT malware evolution by leveraging the information from two sources that complement each other. First, we crawl online articles about IoT malware and employ natural language processing techniques to extract the features of malware samples and their relationships with other malware family, which allow us to form the basic lineage graph. Second, we collect real malware samples through our widely deployed honeypots and design a new classifier to group them into families and identify lineage relationships among them. Such results are used to enhance the basic lineage graph. Eventually, we construct the final lineage graph for 72 IoT malware families by correlating the information from the aforementioned sources, which can help the research community better understand and fight IoT malware now and in the future. Our study has been incorporated into the threat awareness system of NSFOCUS company.**

***Index Terms*—Ensemble learning, evolutionary study, lineage inferring, malware detection, secure Internet of Things (IoT).**

Manuscript received October 16, 2020; revised January 3, 2021 and February 2, 2021; accepted February 28, 2021. Date of publication March 4, 2021; date of current version October 7, 2021. This work was supported in part by the Key-Area Research and Development Program for Guangdong Province under Grant 2019B010136001; in part by the National Key Research and Development Plan under Grant 2017YFB0801801; in part by the National Natural Science Foundation of China (NSFC) under Grant 61672186 and Grant 61872110; and in part by HK RGC Project under Grant PolyU 152239/18E. *(Corresponding author: Hui He.)*

Huanran Wang and Hui He are with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China (e-mail: wanghuanran@hit.edu.cn; hehui@hit.edu.cn).

Weizhe Zhang is with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China, and also with the Cyberspace Security Research Center, Pengcheng Laboratory, Shenzhen 154100, China (e-mail: wzzhang@hit.edu.cn).

Peng Liu is with the College of Information Sciences and Technology, Pennsylvania State University, University Park, PA 16801 USA (e-mail: pliu@ist.psu.edu).

Daniel Xiapu Luo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: csxluo@comp.polyu.edu.hk).

Yang Liu is with the Department of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China.

Jiawei Jiang and Yan Li are with the Department of Computer Science and Technology, Harbin Institute of Technology, Harbin 150000, China.

Xing Zhang, Wenmao Liu, Runzi Zhang, and Xing Lan are with the Innovation Center, NSFOCUS, Inc., Beijing 100089, China.

Digital Object Identifier 10.1109/JIOT.2021.3063840

## I. Introduction

INTERNET-OF-THINGS (IoT) devices have brought a huge impact on people's lives in many ways, such as medical treatment, smart homes, and military research, namely, the smart city [1]. Various studies disclosed that IoT devices mostly use opensource components and frameworks that have numerous security vulnerabilities [2]. Accordingly, various attackers have shifted their attention to those badly secured "smart" devices [3], [4]. After some malware families (e.g., Mirai and Gafgyt) releasing their source codes, IoT malware variants improve their attack performance by adding new configurations, components, and functionalities with direct code reuse from the other opensource families [5]. Unsurprisingly, many variant strains with clear modularization characteristics have emerged, and different opensource families can be easily blended. Compared to traditional malware (such as desktop and mobile malware), IoT malware variants are polymorphic. These variants contain multiple families' features in most cases, which brings challenges for provenance, triage, labeling, lineage analysis, and authorship attribution. To make sense of this complex evolution, the security community has contributed significant findings online [6]–[8] to expose and document these increasing threats [6]–[8]. However, these invaluable findings provide a discursive view for large-scale automated analysis, making it difficult to maintain and scale regularly.

On the academic side, studies are still mostly concentrated on traditional malware detection. However, most of the current studies neglect each family's phylogenetic tree, leading incomplete analysis of malware development trends. For example, some studies only detect a subset of known families [9]–[11]. Some others focus on related areas, such as Android malware [12]–[14], personal computer (PC) virus [15], and Linux malware [16]. Since IoT malware is different from the traditional virus in many aspects, such as the file size, hardware architectures, or even compiler options, therefore, these studies cannot provide an IoT-specific guideline even to malware detection's feature engineering step.

Despite this effort, current IoT malware is still identified on the labels assigned by AV Anti Virus vendors [17], and even if the malware classification was extraordinarily finished, it does not provide any insights about the malware lineage information. Malware lineage in this article refers to the family evolutionary relationships between an original family (e.g., Mirai) and its offsprings, versions, or variants (e.g., Hajime and IoTReaper). In the topic of lineage inferring, existing studies have so far mostly focused on Windows and Android

malware [5], [18], [19]. In most cases, if not all, the prior malware lineage studies tend to consider each input executable a different version [5]. Some other approaches focus on identifying different versions of a given malware family [20]. Therefore, the polymorphic lineage characteristic of the IoT malware ecosystem makes these approaches unsuitable in IoT. Griffioen and Doerr [11] presented a comprehensive study on a large scale of malicious samples and highlighted the tangled relationships of IoT malware families, which can be regarded as a comprehensive theoretical basis for our lineage inferring part. Based on our observation, compared with scientific papers, online technical articles, such as security blogs, company reports, or even malware event news, can offer informative and timely intelligence of malware behavior features and new lineage relationships. A comprehensive study that can integrate existing endeavors is needed due to the unsystematic view of prior studies.

The above discussions about the existing studies demonstrate two real-world challenges: first, a panoramic view of the IoT malware family lineages and feature selection hypotheses are difficult to generate. Second, the traditional approaches cannot meet the requirements of IoT malware classification and lineage inferring. Furthermore, the malware samples for analysis are difficult to collect because each family's exposure time and place are unpredictable.

To address these challenges, we present an evolutionary study with the support of prior intelligence and collected samples, which contains two steps. We employ semantic analysis from IoT malware-related online reports to mine behaviors and lineages between families in the first step. We select these online articles because security companies are more likely to expose their discoveries more timely than academic papers. In the second step, we first implement a honeypot system to collect the real-world active samples to verify the first step further. Then, we propose an ensemble learning-based model for family classification and lineage inference. The analytical results are used to enhance the basic lineage graph generated from the first step. The extensive experiments show that the accuracy of our proposed approach outperforms other security products. Our work presents an evolution study and a solution that can continuously capture active malware in the wild and classify a given sample with the known families and the potential lineage relationship for unknown variants. The well-collected data set can be useful for future research into dependencies and mitigation with the seed data from various technical sources. In summary, our contributions of this work are as follows.

1) We present the first approach for mining the lineages and features from the online literature about IoT malware to form the basic lineage graph.
2) We collect 38 963 IoT malware samples of 36 families through our widely deployed honeypots. To validate our work and motivate further study, we release our data set and feature set to the security community.
3) We implement a practical end-to-end ensemble model for IoT malware classification and lineage analysis. We utilize the analytical result to cross-validate the findings obtained from online articles and construct the final lineage graph for 72 IoT malware families.

The remainder of this article is organized as follows. Section II discusses the relevant studies. Section III presents the motivations of our study. Section IV conducts a comprehensive investigation on the exposed IoT malware families. Section V describes our malware collection, and Section VI designs an ensemble model for malware classification and lineage inferring on real-world samples. Section VII evaluates the performance of the proposed approach. Section VIII discusses the limitations of our work. Section IX summarizes the achievements and future work. We release the full data set and extracted feature set to research[1] for reproducibility purposes. We also share the full lineage graph raw data for ease of extension. Note that our system has already been incorporated into the IoT threat hunt awareness system of NSFOCUS company.[2] Note that we use the term malware throughout the entire paper to refer to IoT malware and the family to IoT malware family to avoid further ambiguity.

## II. Related Work

Many endeavors have been done on the IoT malware landscape and phylogenetic study of variant attacks. We discuss the related works from three aspects according to the emphasis of our study.

### A. Analysis of a Specific IoT Malware Landscape

Researchers have so far mostly focused on analyzing a specific IoT malware family with several case studies. Langner [21] and Durfina *et al.* [22] intensively investigated Stuxnet and Psybot. They discussed their analysis findings and presented several real-world cases to highlight the attack effects, exposed incidents, and social influences of IoT malware. For one of the most notorious Mirai families, Griffioen and Doerr [11] used 7500 IoT honeypots to provide an epidemiologically view into the persistent threat caused by Mirai and its variants. They exploited a flaw in the design of Mirai's random number generator and find that IoT botnets are not self-sustaining.

Some researchers focus on summarizing attack technologies in a survey manner. De Donno *et al.* [23] conducted a taxonomic study of DDoS attacks and mapped the classification to real-world attacks. They then outlineed 14 DDoS-related attacks in IoT and provide a detailed analysis of Mirai. Costin and Zaddach [24] complemented previous studies by gathering additional metrics and meta-information on IoT malware. They released malware with version control and discussed the deficiency in malware identification.

Based on our observation, academic papers are often restricted by limited collection of malware samples, while the industrial community maintains numerous real-time threat discovery platforms. It is unsurprising that the online security reports can offer more detailed and time-sensitive information about specific malware families. Security companies, such as Avast, AWS, and McAfee issue Internet threat analysis reports

[1] https://github.com/IoMafelt/IoMafelt

[2] The Lab focuses on botnet threat identification, DDoS countermeasure technology, and identity authentication security technology for malware detection, provenance, and lineage analysis in their system.

periodically [25]–[62]. Note that the report links cited in this article is part of the information source used in Section IV. The full reference list will be shared along with our data set.

The security community has contributed great endeavors to expose and document the exposed IoT malware families. However, the valuable insights gained from these works provide a disorganized view of the whole IoT ecosystem. In contrast, our work is focused on the whole landscape of IoT malware and seeks to provide a "panoramic" view of the whole landscape.

### B. Malware Detection and Lineage Inferring

Traditional malware detection methods have so far mostly focused on Android malware, Linux desktop malware, and Windows malware, which include signature detection [3], [63]–[65], behavior detection [13], [14], [14], [63], [66]–[71], and taint analysis-based detection [72]–[76]. Recently, machine-learning methods are commonly used for malware classification [9], [15], [77]. Cozzi *et al.* [16] presented the results of the first large scale measurement study, which can greatly help future work in the area of Linux-based malware detection.

While these studies for malware detection provide a valuable guideline for feature engineering, most of them cannot be applied to analyze IoT malware. First, the number of open-source IoT malware families has increased sharply, leading to severe label fragmentation among variants. Only detecting malware in a specific family will lead to misleading situation awareness. Additionally, due to IoT devices' limited computing power, IoT malware will not take the manner of the Trojan style commonly used by traditional viruses (such as Zsone [78]). Therefore, the traditional feature selection and classification strategy are different in the topic of IoT. To make things worse, the existing methods do not consider evasion techniques, such as adversarial inputs, data poisoning attacks, and model stealing techniques.

In the topic of lineage inferring, some existing studies have so far focused on Windows and Android malware. Ruttenberg *et al.* [79] presented a two-stage clustering technology to identify the shared components of Windows 32-bit binaries. They used features constructed from abstracted semantics of basic blocks and grouped similar procedures into shared components using the two-stage clustering technology. In most cases, if not all, the prior malware lineage studies tend to consider each input executable a different version. Studies [5], [80] discover the evolution of a malware family by comparing the code versions and quantifying their differences. Some other approaches focus on identifying different versions of a given malware family [20]. Besides, Cozzi *et al.* [17] presented a code-based clustering method that can report on IoT malware family genealogy. They also track the lineage relationships and conduct large-scale clustering experiments on IoT malware. However, their goal is not to provide a future-proof IoT malware analysis technique. They take advantage of the feasibility of their collected samples, which are currently available for code-based analysis. Their solution is vulnerable to a potential threat that malware developer could easily employ tricks to evade such analysis in the future.

As a remedy, our work aims at filling this gap by Web data mining and ensemble learning. We start with a semantic analysis of online articles, mine malicious behaviors, and mirror them to features. Different from study [17], we quantify the similarity of each family and detect the potential lineages in the system invoke level. While unpacking and deobfuscation can help researchers to extract informative features from attackers' intent, but it is a time-consuming task and automatically dealing with customized packers is still an open task. Therefore, our method does not need any deobfuscation and unpacking process and directly extracts features from the executable file. As a result, our approach can meet the requirements for lineage inferring to incorporate various families' characteristics.

### C. Literature-Based Discovery

Research on mining the semantic information from the prior literature the first proposed in the field of biological science [81]. In other fields, literature mining technology is used to identify genes in the biological field [82]. More recently, Spangler *et al.* [83] combined entity detection with neighbor-text analysis and identified the potential features of entities based on the existing findings. Their approach identified new protein kinases that phosphorylate the protein tumor suppressor p53.

Although NLP technologies have made tremendous progress during the past decade, how to apply NLP technologies to IoT security appropriately is still an open problem. Pandita *et al.* [84] analyzed the description of a given application to bridge the semantic gap between user expectations and application functionality. Zhu and Dumitraş [85] presented an end-to-end approach for automatic feature engineering. They mined scientific information from scientific papers and Android development documentation and mirrored the malware-behavior-feature to the Android-related entity, namely, permissions, intents, and API short for Application Programming Interface calls. Compared to PC security and Smartphone security, IoT security has a unique characteristic, that is, IoT-related blogs and papers rarely mention the specific API or C library used by malware. We fill the gap by using the universal tactics listed by MITRE [86] and manually induce the top-ranked behaviors of each family into five types. Furthermore, we verify and supplement the results of the literature-based approach with a large-scale data set, which can address the limitations of the existing works.

## III. Motivating Questions and Main Challenges

In this work, we present an evolutionary study systematically and holistically in a systematic and a holistic manner by exploring three motivating questions not addressed by the prior work. Note that evolutionary study includes the lineage referring of IoT malware families, track their evolution, classify their variants, and quantify their similarity.

*Q1:* Why do We Select the Informative IoT Malware Features by Mining the Articles Published in the Security Community With Minimum Manual Endeavors?

In feature engineering, security analysts cognitively generate various hypotheses of the common behaviors that appear in

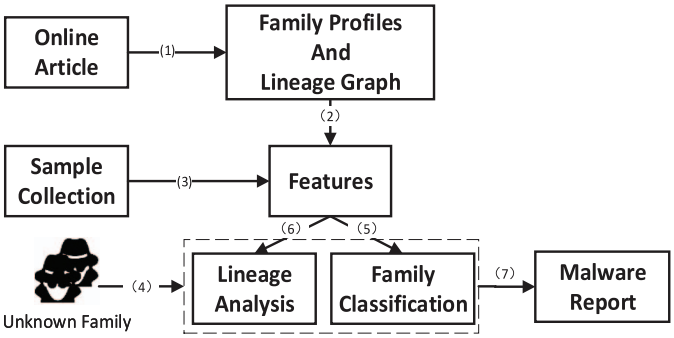



Fig. 1. System framework: (1) intelligence collection (Section IV-A); (2) feature engineering (Section IV-B); (3) static and dynamic analysis (Section VII-B); (4) detectability of new variant (Section VII-C); (5) family classification for real-world malware(Section VI-A); (6) lineage analysis for real-world malware (Section VI-B); and (7) malware report (Section VII-B).

each family [85]. Posteriori analysis is guided by the increasing numerous previous studies that characterize adversary behaviors mirrored to the learning-used features. However, carefully reading thousands of these articles to find a fraction of the relevant knowledge is time consuming and labor intensive. The data-driven alternative approaches reply on the data scale and clean ground truth, challenging to obtain. For instance, VirusTotal detects files by integrating various auto-virus engines, and the results rarely reach unification. Therefore, our work provides an approach for feature selection instead of just relying on our collected data set, which may avoid the biases of the ground truth. Besides, we select online security reports as the intelligence source because security companies are more likely to disclose informative and timely intelligence on online reports instead of scientific papers. This question will be detailed addressed in Section IV-B.

*Q2:* Why do We Automatically Infer Malware Lineage Based on the Existing Literature?

Lineage inferring plays an essential role for sample labeling, family categorization, threat intelligence, and authorship investigation [20], and it can be a rich source of information for various security questions. Each variant inherits the derivative evidence from its original family, and the original IoT malware family is likely not exclusive. Lineage inferring is increasingly challenging in the area of IoT malware. Literature-based lineage inferring is a new approach to fill this gap. This question will be detailed addressed in Section IV-C. Based on the evaluation of our data set, our approach is proven to be capable of verifying known lineages and discovering potential derivative relationships.

*Q3:* Why do We Present Data-Driven Study After Natural Language Processing (Hereafter NLP) Endeavors?

Literature-based study can help establish a panoramic and comprehensive perspective on IoT malware evolution. However, this study is guided by the existed work, thereby indicating that our approach cannot mine the unknown or public unavailable information. Therefore, verification procedures should be presented to ensure that the NLP results are correct and complete. We answer this question by experiments on a large scale of real-world IoT malware samples in Section VII.

*Nongoals:* Honeypots are only used to gradually accumulate active IoT malware samples in our work, and honeypots do not capture the major source of our training set. We do not aim to outperform other existing honeypot technologies. Furthermore, although we have achieved good results using models based on ensemble learning, the learning algorithm and NLP technology are not our main contribution.

To this end, we first collect online articles about IoT malware from security websites. Then, we parse the semantic descriptions and extract the behavior and lineage relationships of IoT malware families by NLP techniques. These descriptions are parsed into characters to outline a temporal family graph. We then map these characters to concrete features that we can automatically extract from samples. Then, the features are used in lineage analysis and family detection. Finally, explanations are generated for why a given variant (although it is a new kind of malware) is detected to have lineages of several families. Fig. 1 illustrates the system architecture of the proposed approach.

## IV. Text Mining of IoT Malware Family Profiles

In this section, to bridge the gap between semantic knowledge and feature engineering, we crawl online articles about IoT malware to form the basic family profiles, which include two aspects of semantic information, malware behaviors (used to generate feature set) and lineage relationships between families (used to form the lineage graph). To ensure malware detection effectiveness, we present data-driven endeavors in Section VII-B for evaluation.

### A. Intelligence Collection

The survey [24] summarized at least 60 IoT malware families, which covers the existed families from 2012 to 2018. We start with an initial keyword list of these family names. We realize that some aliases are missing, leading to an incomplete collection. By searching with the initial keywords on Google, we got 10 214 related results as the initial article base. To expand the alias base, we carefully define the common usages as regular expression patterns in Table I based on the hypothesis that authors tend to introduce a family with its aliases in one sentence. Note that these matching patterns are summarized through numerous manual analysis endeavors from the collected articles. Taking a report from Difesa [87] (search by the keyword "Qbot malware") as an example: "*Qakbot malware (aka Qbot) is evolving*." After matching with the third pattern, "Qakbot" and "Qbot" are extracted. In this case, "Qakbot" is selected as a complement to our alias base.

In this way, 138 family aliases (covering 72 families) are collected. We use these family aliases together with a set phrase "IoT malware/botnet" to search online articles and crawl all the results, including security reports, technical blog posts, and systematization of knowledge papers related to a given family. Then, the duplicate and irrelevant articles are removed. For example, the articles with the phrase "reprinted from" in titles represent duplicates. Finally, 17 857 IoT-malware-related articles are used for analysis. To verify our defined regular expression patterns, we randomly select 2000 articles from seed articles for manual verification. To this end, we extracted the sentences with alias' expression,

TABLE I
REGULAR EXPRESSION PATTERNS FOR ALIAS AND THE HIT NUMBER

| Statement | Hit |
|---|---|
| (?P⟨$alias$⟩\w+),\s***also known as**((?P⟨$alias$⟩\w+)\s*(,\|and)*\s*)+ | 102 |
| (**named**\|**called**)\s+((?P⟨$alias$⟩\w+)\s*\/*\s*)+ | 53 |
| (?P⟨$alias$⟩\w+)\s*,\s***also (named**\|**called**)\s*(?P⟨$alias$⟩\w+) | 319 |
| (?P⟨$alias$⟩\w+)\s*\(**aka**\s+(?P⟨$alias$⟩\w+)) | 408 |
| **known as** (?P⟨$alias$⟩\w+),\s***aka**\s+(?P⟨$alias$⟩\w+) | 73 |
| (?P⟨$alias$⟩\w+)\s*(**alternatively**\s+**called**\s+(?P⟨$alias$⟩\w+)) | 32 |
| (?P⟨$alias$⟩\w+),\s+**siraa patchara**(?P⟨$alias$⟩\w+) | 47 |
| **several**\s+**names,**\s***including**\s+((?P⟨$alias$⟩\w+)(,\s)*)+ | 1 |
| **variously called**\s+((?P⟨$alias$⟩\w+)(,\s)*)+ | 1 |
| **malware will be reffered to as**\s+((?P⟨$alias$⟩\w+)(,\s)*)+ | 1 |
| **goes by the names**\s+((?P⟨$alias$⟩\w+)(,\s)*)+ | 2 |

TABLE II
EXAMPLE OF BEHAVIOR EXTRACTION

| | |
|---|---|
| text | "Mirai is a self-propagating worm, it's a malicious program that replicates itself by finding, attacking and infecting vulnerable IoT devices. Mirai attacks were not tied to a single actor." [89] |
| behavior | Mirai is self-propagating worm<br>Mirai is malicious program<br>Mirai replicate itself<br>Mirai find vulnerable IoT device<br>Mirai attack vulnerable IoT device<br>Mirai infect vulnerable IoT device<br>Mirai attack are not NOT_tied NOT_single NOT_actor. |

namely, the sentences that contain at least two family aliases. Note that in the 2000 test articles, we finally extracted 1039 sentences that meet the requirements. The number of hits for each matched pattern is shown in Table I. Statistics indicate that though the patterns we define may not cover all the possible grammatical usages, these patterns are sufficient to collect the commonly used aliases because of the large number of collected articles.

### B. Malware Feature Mining

Security analysts cognitively generate various hypotheses of the common behaviors that appear in each family, and the reasonable hypotheses about malware behaviors can be used in the feature engineering process [85]. Therefore, malware features can be mined from the content of the existing literature. Based on this, our malware feature mining phase includes two tasks: 1) malware behavior extraction and 2) feature set engineering based on the extracted behaviors. We obtain a high-frequency behavior list of each family in the first task, which is used for feature association in the second task.

*1) Malware Behavior Extraction:* In the task of malware behavior extraction, we define a behavior as a set that includes subject, verb, and object, where either subject or object could be missing. Since single words are not sufficient to provide an unambiguous semantic expression [88], we define behavior as a fundamental primitive in this article. We extract behaviors in two steps: 1) collecting and 2) behavior weighting.

In the first step, we employ the Stanford typed dependency parser [88] to analyze the main components of each sentence and extract behaviors. They defined approximately 50 grammatical relations between a *governor* (also known as a regent) and a dependent. In our work, we present feature inference based on the original expression of behavior. Therefore, we attempt to infer all the grammatical relations between subj, verb, and obj, including the dependencies in a clause. Based on this reason, we apply the following dependency type: *iobj, dobj, nsubj, nsubjpass, rcmod, vmod, xcomp, xsubj*. We take three main steps to reduce the noise in the semantic analysis process. First, if a sentence contains multiple trunks, we complete the sentence with the main clause's subject. Second, to reduce the word variants, we merge words of the same part of speech with similar meanings by WordNet [90]. Third, to eliminate the semantic misunderstanding of negative words, we initially judge whether there are negative words in the sentence based on the list provided by Liu *et al.* [91] and Hu and Liu [92]. We then add the *NOT_* prefix to all words between the negative word and the nearest punctuation. In this way, 62 424 unique behaviors are generated from the 17 857 articles. Table II shows one example of behavior extraction.

In the second step, we collect all the verbs and nouns in the collected behaviors to predict the grammatical relationships between words and family names. Some of our collected articles are only personal opinions or outdated, without enough demonstration, experiments, or peer review. Therefore, directly using this knowledge to build a model may bring unexpected errors. As a remedy, we define the relevance score in (1) between each word and family name based on the method of FeatureSmith [85]. $I(w; \textit{Familyname})$ represents the mutual information between the family name and a given phrase. $H(w)$ in (2) denotes the entropy of phrase $w$. The relevance score for each behavior is the sum of all phrases that make up this behavior. Our processing strategy is based on the hypothesis that those inaccurate statements will not be largely consistent. Many kinds of research based on probabilistic association analysis default to this hypothesis [85]

$$S(w) = \frac{I(w; \text{Familyname})}{H(w)} \tag{1}$$

$$H(w) = -\sum_{i=1}^{i=2} P(w_i)\log_2 P(w_i). \tag{2}$$

To observe the distribution of behavioral purposes, we carefully select all the Linux-related tactics from the system and map the extracted behaviors to a specific tactic for further association with the extractable features. According to our observation, high weight behavior is more frequent in the corresponding family's online articles. Furthermore, the behaviors above 75 rank generally show low document frequency. In this way, each family's top 5400 behaviors are mapped into seven kinds of tactics, which are shown in Table III.

*2) Feature Generation:* Our goal is to map the 5400 behaviors to the features that can be directly extracted from executables in feature set engineering. The features selected for family classification should be cross-sectional, informative, and associated with the extracted behaviors. The literature [16] indicated that the file meta information, segment, system call, and the user commands hard-coded in an executable binary can reflect the behavior characters of different families. We focus on file metadata, system calls, and library functions as potential IoT malware classification features in this work.

The file metadata we used as classification features include binary size, security options during compilation (e.g., Canary and Nx), operating environment [e.g., operating system (OS),

TABLE III
MALWARE BEHAVIOR CATEGORIES

| Behavior Type | Related Family (Partial) | Description | Behavior Percentage |
|---|---|---|---|
| Impact | Xor, Stuxnet, BrickerBot | used to manipulate, interrupt, or destroy system | 12% |
| Persistence | BrickerBot, Mirai, Gafgyt | trying to maintain foothold | 31% |
| Privilege Escalation | Shishiga, SynoLocker | gain higher-level permissions on a device | 23% |
| Defense Evasion | Kaiten, TheMoon, Okiru | avoiding detection throughout their compromise | 10% |
| Credential Access | Raspberry Pi, Ganiw | steal credentials like account names and passwords | 3% |
| Discovery | UPnProxy, Gr1n, NewAidra | gain intelligence about the smart devices | 15% |
| Lateral Movement | Raspberry, Gr1n | enter and control remote devices on a local network | 6% |

TABLE IV
18 SELECTED STATIC FEATURES

| Type | Name | Description | Value Range(Partial) |
|---|---|---|---|
| Basic | Format | File Format | pe,elf,bios,dex,sfc |
| | Type | File Type | exec,dyn,dll,dex,rel,rom |
| | Bits | Target OS Bits | 16, 32, 64 |
| | Humansz | Size of File | - |
| Compile | Arch | System Architecture | x86,ppc,arm,sh,arc,mips |
| | OS | Operating System | win,linux,dos |
| | Lang | Programming Language | c,c++,vb,cil,go,rust |
| | Endian | Data Storage Sequence | little,big |
| | Pcalign | Alignment | 0, 2, 4 |
| Library | Stripped | Dynamic Base | 0, 1 |
| | Static | Static Link | 0, 1 |
| | Relocs | Redirect | 0, 1 |
| | Rpath | Dynamic Link Library | 0, 1 |
| Security | Canary | Stack Overflow Protection | 0, 1 |
| | Nx | No Execute | 0, 1 |
| | Linenum | Line Number Information | 0, 1 |
| | Lsyms | Symbol Table | 0, 1 |
| | Segment | Program headers of ELFs | 0, 1 |

architecture, and memory management unit (MMU) support], and various compilation options. Certain properties in an executable file are needed at runtime by the OSs and are hard to forge. Furthermore, malware authors lack the motivation to fake their programs against these features. Therefore, we select 18 executable binary properties as features for classification, as shown in Table IV.

There are two ways to manipulate files under a Linux-based system: 1) system call and 2) library functions. System calls provide the interface between OS and a process, and they are the only entry points into the kernel system. The library functions are used for application development as application programming interface. As is well known, most of the IoT devices are based on Linux systems, and the malicious behaviors collected in the previous step can be associated with one or more specific system calls or library functions. Therefore, we select system calls and library functions to generate the potential features for IoT malware detection. In total, we have 7422 library functions and 500 system calls as initial features derived from the Linux man-pages project [93]. We associate the behaviors and features in three steps.

The first step is collection. We crawl all the official descriptions of system calls and library functions offered by Linux-man websites. Furthermore, the hyperlinks and formatting tags in HTML are removed, only the descriptions paragraphs are remained.

TABLE V
TOP-FIVE RANKED FEATURES

| Rank | Feature Name | Feature Type |
|---|---|---|
| 1 | getpid | System Calls |
| 2 | getsockname | System Calls |
| 3 | exit | System Calls |
| 4 | fwrite() | Library Functions |
| 5 | getenv() | Library Functions |

The second step is data cleaning. We only keep the verbs and nouns in these descriptions to reduce the noise from irrelevant words because the association between the behavior and the system call is mainly embodied in the entity. Note that to reduce the noise in the semantic analysis process, we also the three-step strategy mentioned in Section IV-B1.

The third step is feature generation. We first define the behavior base $B$ and initial feature set $F$. We use $L$ to refer to the collection size in general. For each $b_i i \in [1, L]$ in $B$, $p_j j \in [1, L]$ denotes the single word or collocation in $b_i$, which is also called "phrase." For each $f_k k \in [1, L]$ in $F$, $w_q q \in [1, L]$ denotes the word filtered from the second step. NLTK [94] presents an API for calculating the distance of two given words. However, to associate $B$ and $F$, we need to calculate the distance between every $b_i$ and $f_k$, which is the simple problem between two words. To solve this problem, we just consider the maximum distance between $p_i$ and $w_{1-L}$ as the distance between $p_i$ and $f_k$. The sum of each $p_i$ is used to represent the distance between $b_i$ and $f_k$. The reason for calculating in this way is to reduce the influence of irrelevant words on distance. The distance from $b_i$ to $f_k$ is calculated using the following:

$$D(b,f) = \sum \max\big[\text{distance}(p_i, w_1), \ldots, \text{distance}(p_i, w_L)\big]. \tag{3}$$

Table V shows the top five features generated in this manner. In total, we obtain 165 system calls and 1278 library functions used as features, which can be easily extracted from the executable files. The *getpid* and *exit* system calls returns the process ID of the calling process. They are often used by routines that generate unique temporary filenames and terminate the calling process immediately. The behaviors that contribute to this feature are also related to operating the running processes, e.g., "kill malware process," "PID in file" (extracted in report [95]). Several botnets malware families laterally expand their scale by scans IP addresses across the Internet. Therefore, the *getsockname* system call is used to return the current address to which the socket *sockfd* is bound. Besides, hackers often present file operators for destructive purposes and add their scripts to the environment list to retain their malicious attempts. Therefore, these behaviors contribute to the features *fwrite*() and *getenv*().

### C. Family Lineage Inferring

Compared with traditional viruses, the IoT malware families evolve more rapidly, the exposed derivation relationships are more intricate, and the variant family number is numerous. Therefore, we employ NLP techniques to extract the

lineage relationships from 17 857 online literature collected in Section IV-A.

To this end, we first remove the sentences that contain less than two names of different families because they are unlikely to express the available lineage-related information. Because there are many words in a sentence, it is impossible to achieve our goal by calculating the word distance. Therefore, we then adopt a simple support vector machine (SVM) model to differentiate two kinds of sentences, namely, the sentences that contain the expression of "lineage" and the sentences without this type of expression. We choose the SVM because it achieves better classification accuracy performance compared to other similar algorithms, such as the decision tree (DT), $K$-nearest neighbor (KNN), etc., when the data size is small, and it is more lightweight for training.

We classify sentences from the perspective of the subordinate clause structure and semantic meaning. Therefore, we adopt the part-of-speech, relative position of word,s and information gain as the training features. We extract part-of-speech and word position by using the *pos_tag* function of NLTK [94]. Information gain is an important indicator used in feature selection, which is used to measure the relevance of a given the word to the meaning of "lineage." We calculate the information gain of a given word by using (5). In this manner, 18 words with high $IG(T)$ are obtained, which are close to the meaning of "lineage" as follows: ["new," "same," "evolution," "likely," "common," "offspring," "later," "derived," "variant," "beyond," "faster," "similar," "merge," "followed," "like," "from," "version," "fusion"]. Then, we employ NLTK to extract the words in the training samples for establishing a thesaurus. In each label $T$, the occurrence probability of each word $C$ in the thesaurus is represented as $P(C)$. Equation (2) is used to calculate the information entropy, which is used to express the uncertainty of information. Conditional entropy embodies the degree of uncertainty after classification by a given feature (4)

$$H(C|T) = P(\text{word})H(C|\text{word}) + P\big(\overline{\text{word}}\big)H\big(C|\overline{\text{word}}\big) \quad (4)$$

$$IG(T) = H(C) - H(C|T). \quad (5)$$

In this way, 9002 sentences are classified as expressing the meaning of "lineage." Then, we extract the family names that appear in every sentence and merge them according to the alias library we obtained in Section IV-A. If the family names identified in the two sentences are the same, they express the same lineage relationship, so we merge them. After filtering, 280 sentences are selected that have the expression of "lineage." After that, we outline the timeline of the families in a lineage graph, and it is completed in Section VII with the cross-validation of real-world samples. Note that this lineage graph is treated as the ground truth based on manual verification.

## V. Data Set: Collecting Comprehensive Set of IoT Samples

Our first goal is to collect a comprehensive and representative malware data set to study IoT malware evolution. To continuously capture active samples in the wide, we design four kinds of honeypots, shown in Table VI.

TABLE VI
FOUR TYPES OF HONEYPOTS

| Simulation Type | Simulation Target (Selected) | Target Family (Selected) |
|---|---|---|
| Embedded Web Service | Goahead (v2.5.1) | Hajime, IoTReaper, BrickerBot, Persirai |
| | BOA (v0.94) | Hajime |
| | AppWeb (v3.6.2) | Unknown |
| | Thttpd (v2.25) | Hajime |
| | Apache (v2.3.1) | Mirai, Gafgyt |
| CVE of Device | CVE-2014-8361 | Satori |
| | CVE-2017-17215 | Mirai, Satori, Okiru |
| | CVE-2017-5174 | Mirai |
| | CVE-2018-10561 | IoTReaper, Gafgyt, Mirai, Satori |
| | CVE-2018-10562 | Mirai, Satori, Gafgyt |
| | CVE-2019-3929 | Mirai |
| General Protocol | SSH | Gafgyt, ProxyM, Shishiga, GoScanSSH |
| | TELNET | Mirai, Shishiga, Aidra, SynoLocker, NyaDrop, Routrem, Amnesia, Gr1n, IoTReaper, Satori, Okiru, OMG, H&S |
| IoT Protocol | UPnP (SSDP, SOAP) | UPnP, UPnProxy, Persirai |

The first type of honeypots is implemented to simulate devices with vulnerable embedded Web services. Embedded Web services refer to the services deployed on embedded IoT devices (such as routers, printers, webcams, etc.), mainly including Goahead, BOA, AppWeb, Thttpd, and Apache. This kind of honeypot can also be used to capture the unknown Common Vulnerabilities and Exposures (hereafter CVEs) due to the multiple Web service. We select a version with an epidemic vulnerability for each Web service, which can be exploited by various IoT families. We deploy the selected version of the services on honeypots and then launch the related services [such as secure shell (SSH), Common Gateway Interface support] to attract attackers. The second type of honeypots is used to simulate popular CVE and deeply focus on specific IoT device version. We implement five CVEs to simulate vulnerable IoT devices in this article. The third type of honeypots is used to capture the IoT malware families that utilize general protocols, namely, SSH and TELNET in this work. To this end, we adopt cowrie [96], an opensourced honeypot framework focus on SSH and Telnet attack capture. The fourth type of honeypots is used to capture the IoT malware families that utilize IoT related protocols. We implement the Universal plug and play (UPnP) honeypots. UPnP is a set of network protocols for home network discovery and connection between devices. To implement UPnP honeypot, we utilize the vulnerability of two protocols, namely, SSDP and SOAP.

Our honeypots are developed on 20 virtual private network (VPN) servers covering 15 countries with dynamic IP addresses. Limited by the honeypot scale and design, most of the captured families are Mirai and Gafgyt. Therefore, we supplement our data set from other sources, and our final malware data set consists of two parts. The first part consists of 16 752 benign samples collected from executable files of Ubuntu Core 18 system AMD 64 bits, RIOT-OS-2020.04, and Contiki-NG-3.0, which are specifically designed for IoT devices. The second part consists of 39 004 IoT malicious samples covering 36 families, which are collected by our developed Cowrie-based honeypots (from the first quarter of 2019 to the first quarter of 2020) and commercial interchange

TABLE VII
DISTRIBUTION OF COLLECTED MALWARE SAMPLES

| Family Name | Alias | Total (ELF/PE) |
|---|---|---|
| Mirai | Mirai.AT, Mirai.backdoor.a | 14,504 (14,504/0) |
| Qbot | Bashlite, Qbot, Ballpit, LizardStresser, Gafgyt, Torlus, Lizkebab, Qakbot, Qakbot, PinkSlip, Pinkslipbot | 10,474 (5,402/5072) |
| RBOT | RBOT, Spybot, Robot | 6,022 (0/6,022) |
| ZLOB | - | 2,791 (0/2,791) |
| AESDDoS | AESDdos, AES, MrBlack, Flooder, Dofloo, Spike, Wrkatk, Sotdas, | 1,248 (1,192/56) |
| Aidra | LightAidra, Zendran | 793 (793/0) |
| Kaiten | Sunami, Sunam, Tsunami | 748 (745/3) |
| DNSChanger | - | 554 (0/554) |
| Ganiw | - | 406 (406/0) |
| Reverseshell | Small, Bashlet | 318 (31/287) |
| Xor | XorDDoS | 230 (230/0) |
| VPNFilter | - | 180 (180/0) |
| Amnesia | - | 137 (32/105) |
| Elknot | Billgates, Mayday, Chikdos | 115 (71/44) |
| Stuxnet | - | 111 (0/111) |
| Persirai | - | 67 (67/0) |
| Chalubo | ChaCha-Lua-bot | 62 (62/0) |
| Hydra | - | 54 (0/54) |
| GoScanSSH | - | 51 (51/0) |
| Hajime | - | 35 (35/0) |
| Coinminer | - | 21 (11/10) |
| OMG | OMG! | 26 (0/26) |
| Bashdoor | Shellshock,Shell.shock,Shell shock | 16 (16/0) |
| Knight | - | 9 (9/0) |
| IoTroop | IoTReaper,Reaper | 7 (7/0) |
| JenX | Jen,Jennifer | 6 (1/5) |
| Conficker | - | 5 (5/0) |
| PNScan | Pinscan | 3 (3/0) |
| LuaBot | - | 2 (2/0) |
| SORA | - | 2 (2/0) |
| WICKED | - | 2 (2/0) |
| MIORI | - | 1 (1/0) |
| Muhstik | - | 1 (1/0) |
| Psyb0t | Psybot | 1 (0/1) |
| Satori | - | 1 (1/0) |
| NewAidra | - | 1 (1/0) |

(from the third quarter of 2012 to the third quarter of 2019). Table VII shows the data distribution of two parts.

Note that some families are relatively rare and have fewer samples (less than 10). We do not consider such families in training. In total, 38 963 IoT malware samples of 23 families are used for training. As a comparison, the largest measurement study to date was performed on 93 652 samples [17]. The family labels are determined by AVClass [97]. It is a great idea to label the traditional malware. However, the engines in VirusTotal are not customized for IoT, which has caused many IoT malware samples to be labeled as traditional malware families. For example, we retrieved the VirusTotal report of a Mirai sample.[3] There are only 33 engines can successfully identify it as "malicious" (75 in total) and only 12 of them were identified as "Mirai related." We are aware of this labeling issue in IoT malware detection. Therefore, the labels of our data set are determined by thorough tremendous-effort-involving, and all the used samples are publicly tested between vendors, which addresses this concern to some extent. To foster research in the area of IoT malware detection and to enable a comparison of future approaches, we make the samples collected in our work as all extracted feature sets available to other researchers under https://github.com/IoMafelt/IoMafelt.

[3]MD5: 1da3d6d1a1b0d7ca83e37ac4bd77fdd1.

TABLE VIII
DISTRIBUTION OF THE COLLECTED MALWARE SAMPLE ACROSS MULTIPLE ARCHITECTURES

| Architecture | Number |
|---|---|
| i386 | 15,225 |
| ARM 32Bits | 7,970 |
| Intel 80386 | 4,882 |
| MIPS 32Bits | 4,351 |
| Sparc SUN | 1,372 |
| AMD 64Bits | 1,238 |
| Hitachi SH | 1,035 |
| Motorola M68K | 1,080 |
| MIPS 64Bits | 76 |
| ARC 16Bits | 12 |

### A. Feature Distribution of Samples

The detailed statistics about the distribution of the malware samples in our data set is shown in Appendix A. Due to the space limitation, we discuss the diversity and the significant differences in features selected by NLP to help the research community better understand and fight IoT malware now and in the future.

*Architecture:* Table VIII shows the architectural distribution among the samples, which can provide a guideline of the commonly used architectures for honeypots.

*Size:* Some families have hardcoded weak password lists embedded in their binary files, resulting in a large file size, such as most Mirai samples. Based on our observation, most malware authors attempt to import dynamic system libraries to reduce the file size. Some families, such as Gafgyt, have opened their functions as public APIs to achieve additional functions. The malware opens the *sockprintf* interface to feedback the status information to the server. Therefore, the file size of Gafgyt is generally larger than other families.

*Bits:* Our data set has only 1381 64-bits executable files. This distribution can also coincide with the characteristics of IoT devices. Namely, they are mostly lightweight embedded devices that lack security mechanisms for domain-specific computations.

*System Call:* The system calls are also informative for malware classification. This study takes Linux-based malware as an example. Many families obtain the hostname by using a system call named *gethostname*. Gafgyt obtains the process id list by *getpid* and invokes *exit* to terminate the specified process. BrickerBot deletes system files by *rmdir*.

*Packer:* The automatic recognition of packers remains a subtle topic, and the segment partition of a binary file is affected by the packer programs. We can try to infer whether an executable is packed from segment name, file entropy, and the number of functions [16].

### B. Feature Extraction

We implement a simple sandbox based on an opensource project, called Detux [98], to extract the features selected by NLP in Section IV. We use a QEMU hypervisor to emulate Linux for various CPU architectures based on our samples' architectural distribution.

To ensure that the samples are correctly executed, we install *uClibc*, *musl*, and *glibc* in different analytical environments in

TABLE IX
DESIGN OF FOUR KINDS OF WEAK MODELS

| Name | Training Mode | Model Number | Description |
|---|---|---|---|
| Total Model | $\{T(\{f_1, f_2, ..., f_i, ..., f_N\}, N)\}$ | 1 | This multi-class model classifies from a macro perspective, which focuses on improving the overall accuracy. |
| Simple Model | $\{T(\{f_i, A - f_i\}, 2) \mid i \in [1, N]\}$ | $N$ | This model is dichotomous and focuses on the differences between a given family with all of the other families. |
| Double Model | $\{T(\{f_i, f_j, A - F_i - f_j \mid i \neq j \wedge i, j \in [1, N]\})\}$ | $\frac{N*(N-1)}{2}$ | This model has three labels, making a fine-grained classification by offsetting the bias between multiple families. |
| Black-White Model | $T(Malware, BenignSamples, 2)$ | 1 | This model is used to detect a sample benign or not. |

[1] . $N$ represents the total number of families; $T(s, n)$:$T$ represents an n-class model, and the training set is $s$; $f_i(i \in [1, N])$ represents $i$th family; $A$ represents the total family set.

advance. After these initial steps, we present a 10-min execution for each sample. During the runtime, we trace and obtain the invoked system calls. After the dynamic analysis, we extract the static features in Table IV by r2dare [99].

## VI. FAMILY CLASSIFICATION AND LINEAGE ANALYSIS

In this section, we perform two aspects of analysis: 1) IoT malware family classification and 2) potential lineage identification of malware. The features used for the two kinds of detection are discussed in Section V.

### A. Family Classification for Real-World Malware

Traditional detection based on manual reverse engineering cannot cope with the evasion techniques, such as adversarial inputs, data poisoning attacks, and model stealing techniques. Ensemble learning uses multiple basic learning algorithms (as known as weak models) to obtain better classification performances than that could be obtained from any of the weak constituent models [100]. Therefore, we adopt ensemble learning, consisting of multiple simple models, namely, weak models. Ensemble learning is based on a hypothesis: when the weak models are correctly combined, we can get a more accurate and more robust model [101].

The design of the weak models is shown in Table IX. We initiate a tradeoff between accuracy and efficiency and propose four types of weak models, namely, *double*, *simple*, *black-white*, and *total* models.

Note that the benign samples are investigated in this work for two reasons. First, a typical application scenario for malware analysis is to distinguish whether a sample is benign or malicious. Second, the real samples used for malware classification will contain benign executable files, such as system binaries. Therefore, by considering benign samples, our work's applicability and practicality can be improved, and the noisy benign samples can be removed by our proposed *black-white* model to improve the detection rate.

In this work, we classify 23 collected families for model training, and the total number of involved weak models is 253. The final detection result is decided by a linear additive weighted vote of each model. To derive each weak mode's weight, we first separately conduct a 10-fold cross-validation test on the collected data set for each weak model. Then, we select $f$-measure as the weight of vote for the ensemble model. Note that $f$-measure is the harmonic mean of precision and recall, which can reflect the model's reliability for each possible result. During testing, the black–white model, is first used to detect a given sample is benign or malicious. After that, the other weak models are simultaneously conducted for further detection. Taking the XOR-*Mirai-Others* model (one of the *double* models) as an example, this model is used to identify that a given sample is XOR, *Mirai* or *Others*. If the detection result is XOR, and the corresponding $f$-measure of this model we obtain from the training process is 87%, then the weight of this model, in this case, is 0.87. After that, we add up the weight of all weak models of the double model. In this way, we obtain the vote value of all the weak models for each family. The family with the highest vote value is used as the final classification result. Heterogeneous weak models are targeting different recognition aspects can improve the robustness of the entire ensemble model since every intention to bypass the ensemble model may affect the output of each weak model, thereby immensely increasing the attack cost [100].

### B. Lineage Analysis for Real-World Malware

A daunting challenge faced by malware lineage analysis is the difficulty to validate the analytical results. We propose to take a cross-validation approach to address this challenge. Specifically, our idea is to let the results obtained from NLP-based lineage analysis and sample-based analysis enhance each other. As discussed in Section IV-B, system calls are correlated with malware behaviors. The main malicious characteristics are still retained after the malware is improved by variants [80]. This finding indicates that the system calls can detect the derivative lineage. Furthermore, even for new variants, their malicious intentions are similar to the original ones. Therefore, new variants can be effectively identified through the system call, verified in Section VII-C.

For the sake of convenience, we process each feature as word vectors and obtain the top-ranked frequent vectors (called frequent strings) of each family. Given a sample $S$, we then compare each extracted feature $s$ with all the frequent strings of each family $f$. To measure the lineage purity of each family, seven variables are defined in Table X.

The true prevalence factor $R_{sf}$ in (6) is used to represent the existence probability of $s$ in all the families except $f$. Subsequently, we define the quality factor in (7), $Q_{sf}$ to represent the uniqueness of string $s$ in family $f$

$$R_{sf} = \frac{n * p_s - m_f * p_{sf}}{n} \tag{6}$$

$$Q_{sf} = \frac{p_{sf}}{R_{sf}}. \tag{7}$$

TABLE X
SEVERAL IMPORTANT VARIABLE DEFINITIONS

| Name | Formula | Value Range |
|---|---|---|
| string number of S | $T$ | - |
| whether S contains the $s$ | $x_s$ | 1, -1 |
| $s$ is frequent to $f$ | $y_{sf}$ | 1, 0 |
| sample number of $f$ | $m_f$ | - |
| sample number | $n$ | 38, 963 |
| frequent string number of $f$ | $k_f$ | - |
| infrequent string number of $f$ | $t_f$ | - |
| probability of $s$ exists in $f$ | $p_{sf}$ | - |
| probability of $s$ exists in total | $p_s$ | - |

Thereafter, we define mainstream quality factor $N_f$ in (8) and nonmainstream quality factor $N_f$ in (9) to represent the feature expression ability of the frequent and infrequent strings, respectively, in family $f$

$$M_f = \frac{1}{k_f} * \sum_{i=0}^{|T|} \left(x_i * y_{sf} * Q_{sf}\right) \tag{8}$$

$$N_f = \frac{1}{t_f} * \sum_{i=0}^{|T|} \left[x_i * \left(y_{sf} - 1\right) * Q_{sf}\right]. \tag{9}$$

To compare and weight features of different magnitudes, $M'_f$ and $U'_f$ are normalized in [0, 2000]. Derivative lineage factor $L_f$ is defined in (10) and used to represent the derivative lineage purity of family $f$ that $S$ belongs

$$L_f = M'_f + U'_f. \tag{10}$$

Finally, we rank $L_f$ of each family and select the top three as the derivative lineage analysis results. We present a set of experiments and a case study in Section VII to verify the correctness of the results.

## VII. RESULTS AND FINDINGS

In this section, we first introduce the experimental setup of evaluation and address four research questions. Subsequently, we present a case study in Section VII-C to evaluate the detectability for lineage inferring.

*RQ 1:* Which classifier is appropriate for malware classification? How is the performance of our work compared with other security products in terms of IoT malware detection?

*RQ 2:* Can we improve the results of feature engineering in Section IV-B by feature selection?

*RQ 3:* Can our lineage inferring approach in Section VI verify the existing lineages and supply new ones with our data set?

*RQ 4:* What other trends and key insights can be acquired by profoundly analyzing the malware samples?

### A. Experimental Setup

A 10-fold cross-validation is adopted to estimate the performance of the proposed technique and prevent overfitting. We perform the cross-validations in parallel, and each of the subsets is used once for validation. All experiments are run on an Intel Xeon CPU E5-2667 v4 @ 3.20 GHz with 503 GB of memory running *Ubuntu 14.04.4*. Table XI lists the metrics used in the evaluation.

TABLE XI
DESCRIPTIONS OF USED PERFORMANCE METRICS

| Metrics | Description |
|---|---|
| TP | Malware in family $f$ are correctly classified into $f$. |
| TN | Malware not in family $f$ are incorrectly classified. |
| FN | Malware in family $f$ are not classified into $f$. |
| FP | Malware not in family $f$ are correctly classified. |
| $R_{micro}$ | $\bar{TP}/(\bar{TP} + \bar{FN})$ |
| $P_{micro}$ | $\bar{TP}/(\bar{TP} + \bar{FP})$ |
| $F_{micro}$ | $2 * P_{micro} * R_{micro}/(P_{micro} + R_{micro})$ |
| Mutual Information | $I(X;Y) = H(X) - H(X\|Y)$ |

TABLE XII
COMPARISON BETWEEN DIFFERENT ML ALGORITHMS

| Training Type | ML Algorithm | $R_{micro}$ | $P_{micro}$ | $F_{micro}$ |
|---|---|---|---|---|
| Black-White Model | **GBDT** | 0.99 | 0.98 | **0.98** |
| | NB | 0.84 | 0.83 | 0.83 |
| | SVM | 0.94 | 0.96 | 0.95 |
| | LR | 0.79 | 0.77 | 0.78 |
| | KNN | 0.96 | 0.95 | 0.95 |
| Simple Model | **GBDT** | 0.79 | 0.84 | **0.81** |
| | NB | 0.54 | 0.63 | 0.58 |
| | SVM | 0.81 | 0.45 | 0.57 |
| | LR | 0.41 | 0.52 | 0.46 |
| | KNN | 0.77 | 0.76 | 0.76 |
| Double Model | **GBDT** | 0.91 | 0.92 | **0.91** |
| | NB | 0.62 | 0.53 | 0.57 |
| | SVM | 0.77 | 0.69 | 0.73 |
| | LR | 0.26 | 0.44 | 0.33 |
| | KNN | 0.88 | 0.84 | 0.86 |
| Total Model | **GBDT** | 0.81 | 0.79 | **0.80** |
| | NB | 0.60 | 0.50 | 0.54 |
| | SVM | 0.69 | 0.68 | 0.68 |
| | LR | 0.48 | 0.54 | 0.51 |
| | KNN | 0.72 | 0.49 | 0.58 |
| Ensemble Model | **GBDT** | 0.95 | 0.96 | **0.95** |
| | NB | 0.65 | 0.55 | 0.60 |
| | SVM | 0.84 | 0.71 | 0.77 |
| | LR | 0.53 | 0.57 | 0.55 |
| | KNN | 0.80 | 0.75 | 0.77 |

### B. Evaluation of Approach Capability

*Experiment 1 (RQ1):* We first evaluate the weak models by adopting different classifiers, including gradient boosting DT (GBDT), Naive Bayes (NB), SVM, logistic regression (LR), and KNN, to choose the appropriate classifier for detection. In total, 38 963 samples of 23 families are used for evaluation.

We use the grid search method [102] to select the optimal value of each parameter for each model. Finally, each model gets an optimal value combination of parameters, and the comparison results of different algorithms are presented in Table XII. Based on the comparison, we select GBDT as the default classifier because of its superior performance.

We compare the detection accuracies of the four types of weak models and ensemble model by GBDT. The results are shown in Table XIII. The ensemble model outperforms the four types of weak models because the design of multiple weak models balances the whole model's detection capabilities among some similar families (such as Aidra, Hydra, Gafgyt, and Mirai). After integrating the weak models into the ensemble model, the weak models of low accuracy are given low weights to improve each family's overall accuracy. Besides, we notice that ZLOB, Chalubo, and DNSChanger achieve low detection accuracy. We analyze the detection results and found that there are many false positives between DNSChanger and ZLOB, which means the two families are very similar in terms of the features we selected. We speculate that there is a lineage

TABLE XIII
DETAILED RESULT OF ENSEMBLE MODEL

| Family | B&W Model | | | Simple Model | | | Double Model | | | Total Model | | | Ensemble Model | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R | P | F | R | P | F | R | P | F | R | P | F | R | P | F |
| Mirai | - | - | - | 0.80 | 0.75 | 0.77 | 0.69 | 0.70 | 0.69 | 0.35 | 0.68 | 0.46 | 0.86 | 0.81 | 0.83 |
| Qbot | - | - | - | 0.85 | 0.88 | 0.86 | 0.90 | 0.87 | 0.92 | 0.72 | 0.69 | 0.70 | 0.89 | 0.88 | 0.88 |
| RBOT | - | - | - | 0.98 | 0.98 | 0.98 | 0.95 | 0.96 | 0.95 | 0.97 | 0.90 | 0.93 | 0.96 | 0.96 | 0.96 |
| ZLOB | - | - | - | 0.72 | 0.61 | 0.66 | 0.89 | 0.92 | 0.90 | 0.85 | 0.80 | 0.83 | 0.75 | 0.75 | 0.75 |
| AESDDoS | - | - | - | 0.73 | 0.87 | 0.79 | 0.85 | 0.92 | 0.88 | 0.68 | 0.65 | 0.66 | 0.94 | 0.98 | 0.96 |
| Aidra | - | - | - | 0.75 | 0.83 | 0.79 | 0.86 | 0.91 | 0.88 | 0.72 | 0.67 | 0.80 | 0.91 | 0.95 | 0.93 |
| Kaiten | - | - | - | 0.67 | 0.75 | 0.71 | 0.86 | 0.84 | 0.85 | 0.95 | 0.84 | 0.89 | 0.96 | 0.93 | 0.94 |
| DNSChanger | - | - | - | 0.51 | 0.46 | 0.48 | 0.68 | 0.82 | 0.74 | 0.26 | 0.31 | 0.28 | 0.67 | 0.80 | 0.73 |
| Ganiw | - | - | - | 0.91 | 0.96 | 0.94 | 0.98 | 0.94 | 0.96 | 0.89 | 0.70 | 0.78 | 0.95 | 0.98 | 0.96 |
| Reverseshell | - | - | - | 0.91 | 0.66 | 0.77 | 0.87 | 0.94 | 0.90 | 0.95 | 0.97 | 0.96 | 0.94 | 0.97 | 0.95 |
| Xor | - | - | - | 0.74 | 0.75 | 0.74 | 0.89 | 0.96 | 0.92 | 0.90 | 0.94 | 0.92 | 0.98 | 0.95 | 0.96 |
| VPNFilter | - | - | - | 0.90 | 0.96 | 0.93 | 0.84 | 0.75 | 0.79 | 0.92 | 0.78 | 0.84 | 0.93 | 0.95 | 0.94 |
| Amnesia | - | - | - | 0.40 | 0.85 | 0.54 | 0.92 | 0.95 | 0.93 | 0.89 | 0.82 | 0.85 | 0.93 | 0.94 | 0.94 |
| Elknot | - | - | - | 0.96 | 0.87 | 0.91 | 0.96 | 0.95 | 0.95 | 0.80 | 0.91 | 0.85 | 0.97 | 0.96 | 0.96 |
| Stuxnet | - | - | - | 0.92 | 0.93 | 0.92 | 0.95 | 0.97 | 0.96 | 0.82 | 0.83 | 0.82 | 0.96 | 0.98 | 0.97 |
| Persirai | - | - | - | 0.95 | 0.87 | 0.91 | 0.77 | 0.80 | 0.78 | 0.70 | 0.73 | 0.71 | 0.92 | 0.90 | 0.91 |
| Chalubo | - | - | - | 0.32 | 0.44 | 0.37 | 0.51 | 0.42 | 0.46 | 0.33 | 0.33 | 0.33 | 0.61 | 0.59 | 0.60 |
| Hydra | - | - | - | 0.94 | 0.80 | 0.86 | 0.95 | 0.97 | 0.96 | 0.89 | 0.86 | 0.87 | 0.96 | 0.95 | 0.95 |
| GoScanSSH | - | - | - | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 |
| Hajime | - | - | - | 0.75 | 0.72 | 0.73 | 0.94 | 0.84 | 0.89 | 0.62 | 0.42 | 0.5 | 0.92 | 0.98 | 0.95 |
| Coinminer | - | - | - | 0.90 | 0.73 | 0.81 | 0.98 | 0.98 | 0.98 | 0.90 | 0.91 | 0.91 | 0.97 | 0.98 | 0.97 |
| OMG | - | - | - | 0.98 | 0.98 | 0.98 | 0.96 | 0.90 | 0.93 | 0.88 | 0.87 | 0.87 | 0.97 | 0.96 | 0.97 |
| Bashdoor | - | - | - | 0.93 | 0.87 | 0.90 | 0.95 | 0.96 | 0.95 | 0.78 | 0.70 | 0.74 | 0.98 | 0.96 | 0.97 |
| Benign | 0.99 | 0.98 | 0.98 | - | - | - | - | - | - | - | - | - | 0.99 | 0.98 | 0.98 |
| Average | 0.99 | 0.98 | 0.98 | 0.79 | 0.84 | 0.81 | 0.91 | 0.92 | 0.91 | 0.81 | 0.79 | 0.80 | **0.95** | **0.96** | **0.95** |

TABLE XIV
COMPARISON OF DETECTION ACCURACY WITH OTHER PRODUCTS

| Family Name | Sample | Products | | | Our Work |
|---|---|---|---|---|---|
| | | F-Secure | Avast | McAfee | |
| Mirai | 7,800 | 0.89 | 0.95 | 0.55 | 0.97 |
| Qbot | 3,050 | 0.88 | 0.97 | 0.50 | 0.93 |
| Aidra | 150 | 0.98 | 0.67 | 0.88 | 0.97 |
| DNSChanger | 22 | 0.68 | 0.59 | 0.27 | 0.77 |
| Total | 11,022 | 0.88 | 0.95 | 0.54 | 0.95 |

TABLE XV
FIVE MOST INFORMATIVE FEATURES

| Feature | Mutual Information | Type |
|---|---|---|
| getpid | 0.31 | System Calls |
| getsockname | 0.28 | System Calls |
| size | 0.25 | Metadata |
| gethostname | 0.22 | System Calls |
| getenv() | 0.21 | Library Functions |

relationship between them, which is confirmed in experiment 3. In the case of Chalubo, we find that Chalubo malware reuses a few code snippets from Mirai by reverse engineering, causing the low detection accuracy.

Since we are the first IoT malware classification research toward 23 malware families, we compare our work's detection performance with other security products offered by VirusTotal (e.g., McAfee, Avast, and F-Secure) to verify the effectiveness horizontally. We retrieve the VirusTotal reports of each sample by MD5. We then process them with AVClass [97] to determine the result of each engine.

To perform a fair comparison of the feature sets, we select the samples collected by honeypots (remove the families with less than 50 samples) and 16 752 benign samples as the test set. Note that we choose these honeypot samples for this experiment because most of them are active, and the result could be timely and informative to the security community. We detect whether a given sample is malicious or not because these engines are designed for any files, such as Linux-like, Android, PC malware, or even malicious PDFs, which could bring misinformation. Table XIV shows the comparison result of our work and other products.

Table XIV shows that our study outperforms others in terms of the detection rate. The result shows that the three products are relatively good at detecting hot families (e.g., Mirai and Gafgyt), partly because they tend to misreport other families, such as Mirai and Qbot. We speculate that these products' low detection rates are because of the lack of signature in their base. The detection of each sample consumes 20 s on average, including feature extraction.

*Experiment 2 (RQ2):* To answer RQ2, we evaluate the contribution of individual features to the model's performance by the mutual information metric [103]. Intuitively, mutual information quantifies the loss of uncertainty for malware detection when the executable file has the given feature. We rank the features by calculating the mutual information between every feature and family. Our feature set includes file metadata, system calls, and library functions. Table XV shows the top-five features.

After the feature selection, 146 features are selected from the result of NLP-based feature generation in Section IV-B2, including 18 file metadata features, 72 system calls, and 56 library functions. These removed features are not representative of the families covered by our data set. However, we also find some potential informative behaviors. For example, approximately 20% of the samples are packed. The binaries' segments are interpolated by the pack programs with a random name, which are likely to be parsed by rules, such as the same prefix or the segment name's length. These features will be released for the researchers interested in binary analysis.

*Experiment 3 (RQ3):* To answer RQ3, we evaluate the detectability of our lineage inferring approach with all the samples in our collection. Table XVI shows the coverage accuracy

TABLE XVI
COVERAGE ACCURACY BASED ON THE TOP FIVE CLUSTERS

| Family | TOP | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| Cover Accuracy | 0.84 | 0.95 | 0.98 | 0.98 | 0.98 |

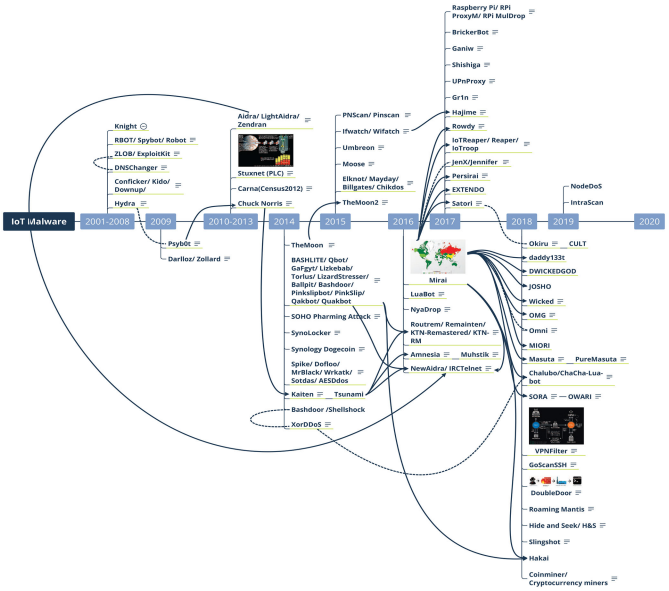



Fig. 2. Batch lineage inferring among all collected families.

of the top $X$ ($X \in [1, 5]$) families ranked by the derivative lineage factor. We determine the analytical results by comparing whether the top $X$ families cover the ground-truth or not.

The result shows that when $X$ changes from two to three, the accuracy significantly increases and flattens out after 3, indicating that the derivative lineage factor is feasible as the metric for lineage inferring. In lineage analysis, a given sample contains the families' characteristics that scored in the top 3. Therefore, the families with a low score contribute little to lineage analysis. In this way, given a family $A$, we can infer whether a lineage relationship exists between $A$ and another family $B$ by the average score among samples.

The results are shown in Fig. 2. We use $F[1-23]$ to represent families with the same order with Table VII due to the page limit. We positively map the score to the pixel value. The result is consistent with the inference based on the literature in Section IV. According to our observation, the distribution of the scores is polarized. We selected 1600 as the threshold based on the scores. Only if the relationships are with a score higher than the threshold, we believe it is existing. With this hypothesis, we discover five new lineages and verify 12 lineages obtained by Section IV (Table XVII). After this experiment, we complete the lineage graph in Fig. 3.

### C. Case Study (RQ4)

*Example 1 (Detectability of NewAidra):* NewAidra, also called IRCTelnet, is a combination of Mirai, Aidra, and BASHLITE. However, the *NewAidra* samples collected in our data are few. Therefore, this family is removed from the training set and becomes "unknown" to us.

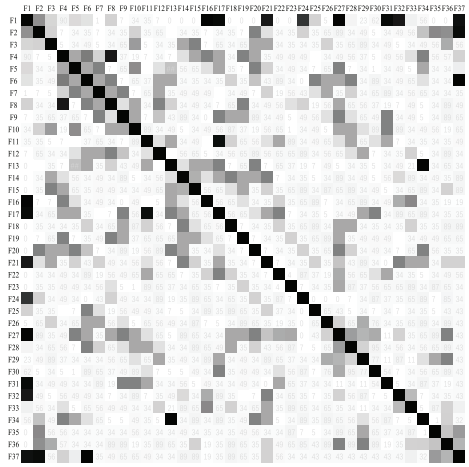


Fig. 3. IoT Malware Family Derivative Relationships. Each branch represents a malware family. Different aliases of the same family are separated by "/." The directed lines' starting points represent the original families, whereas the ending points represent the derivative families. A derivative family will be categorized as a child node when exposed in the same year with the original family. The solid line represents the derivative relationship that can be mined from online articles. The imaginary line represents the derivate relationship that can be inferred from our data set. We used "*" to indicate the families with insufficient samples collected until the fourth quarter of 2019.

TABLE XVII
CONTRIBUTIONS FOR LINEAGE RELATIONSHIPS BY SAMPLE ANALYSIS

| Lineage Relationship | Derivative Lineage Factor | Contribution |
|---|---|---|
| DNSChanger-ZLOB | 1870.5 | New Discovery |
| Chalubo-Xor | 1702.3 | New Discovery |
| JeX-Mirai | 1902.1 | New Discovery |
| Xor-Bashdoor | 1690.2 | New Discovery |
| Psyb0t-Hydra | 1765.0 | New Discovery |
| Mirai-Chalubo | 1977.3 | Verification |
| Mirai-Persirai | 1905.0 | Verification |
| Mirai-OMG | 1742.8 | Verification |
| Mirai-Wicked | 1602.0 | Verification |
| Mirai-Sora | 1800.5 | Verification |
| Mirai-Satori | 1623.0 | Verification |
| Mirai-IoTReaper | 1765.0 | Verification |
| Mirai-MIORI | 1790.3 | Verification |
| Amnesia-Muhstik | 1844.0 | Verification |
| Aidra-NewAidra | 1950.8 | Verification |
| Qbot-NewAidra | 1661.5 | Verification |
| Mirai-NewAidra | 1720.1 | Verification |

```
RemoveTempDirs();
v28 = mainCommSock;
v29 = get_telstate_host(v61 + 28 * i);
sockprintf(
  v28,
  "[ GALAXY ] Removing Temp Directorys. || IP: %s || Port: 23 || Username: %s || Password: %s",
  v29,
  Telnet_Usernames[*(unsigned __int8 *)(28 * i + v61 + 10)],
  *(&Telnet_Passwords + *(unsigned __int8 *)(28 * i + v61 + 11)));
sprintf(&v48, "pkill -9 %s;killall -9 %s;", Bot_Killer_Binarys, Bot_Killer_Binarys);
```

Fig. 4. SockPrint function in $X$.

We select the sample[4] of this family to present a case study for evaluating the detectability of lineage.

First, we extracted the static features of the selected sample (hereinafter referred to as $X$). We determine that $X$ is designed for multiarchitecture IoT devices. Besides, a telnet protocol-based scanner tool is built-in. The function *sockprint()* (shown in Fig. 4) of $X$ is first used in the source code of Gafgyt. It is used to maintain a channel with remote server. $X$ also uses a

[4]MD5: 9d06edc77db2558adb90f942b187c9d2.

```
v15 = *(_DWORD *)(28 * i + v62);
v16 = Mirai_Usernames[*(unsigned __int8 *)(28 * i + v63 + 10)];
v17 = strlen(Mirai Usernames[*(unsigned __int8)*)(28 * i + v63 + 10)]);
if ( send(v15, v16, v17, 0x4000) >= 0 )
{
  if ( send(*(_DWORD *)(28 * i + v62), "\r\n", 2, 0x4000) >= 0 )
    advance_telstate(v62 + 28 * i, 4);
  else
    reset_telstate(v62 + 28 * i);
}
```

Fig. 5. Scan function of *Mirai* in *X*.

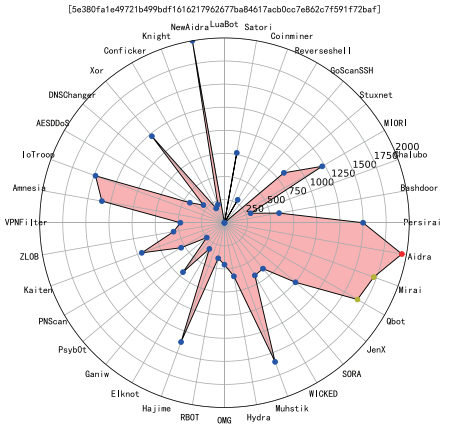



Fig. 6. Lineage analysis result of *X*.

leaked credential list, which is copied from *Mirai*, as shown in Fig. 5.

Furthermore, we observed that *X* could kill running instances by itself and download the latest available version. It checks the status of the process before executing an attack. *X* obtains the system information of the compromised device and sends an attack request to the remote server to start subsequent Internet relay chat (IRC) connections.

We used *X* as an input to our developed system. The family detection result is *Aidra*. In the derivative lineage analysis, we obtained the top three derivative lineage factors, namely, *Aidra* (1950.8), *Mirai* (1720.1), and *Qbot* (1661.5). The radar chart in Fig. 6 consists of a sequence of equiangular spokes, called radii, with each spoke representing the lineage similarity between *X* and each family. Our method rightfully recognizes them as belonging to the same family but as distinct variants.

*Example 2 [Mirai (the Largest Family)]:* Mirai is one of the most notorious malwares in history, whose samples account for almost 37% of our data set. Its multiple version codes are available online, which breed a continuous proliferation of emerging variants. Mirai utilizes the hard-coded command (e.g., *cat /proc/version, cat /proc/cpuinfo*) to obtain knowledge about smart devices. According to our observation, Mirai is the family with the largest number of variants. The two main types of modifications to Mirai are as follows: the first type of variants focuses on the spread, including ADBMiner, Okiru, and Wicked; the second one focuses on function customization. Attackers redevelop the source code to remove or improve some functions. For example, OMG and JenX use the opensource toolkit 3proxy to remove the scan component in Mirai.

```
if ( send(
       *(_DWORD *)(28 * i + v35),
       "cd /tmp; wget http://catsmeowalot.com/a; chmod 777 a;
       267,
       0x4000) >= 0 )
{
```

Downloader

```
if ( !(v27 | v28) == v27 )
{
  system("sudo yum install python-paramiko -y;sudo apt-get install python-paramiko -y;
  ClearHistory();
  sockprintf(mainCommSock, "Installing Python Scanner");
}
```

Scanner

```
if ( result > 0 )
{
  result = listFork( );
  if ( !result )
  {
    v78 = atol(*(_DWORD *)(a2 + 24));
    v79 = atol(*(_DWORD *)(a2 + 20));
    v80 = *(_DWORD *)(a2 + 16);
    v81 = atol(*(_DWORD *)(a2 + 12));
    SendHTTP(*(_DWORD *)(a2 + 4), *(_DWORD *)(a2 + 8), v81, v80, v79, v78);
    exit(0);
  }
}
```

DDoS Module

Fig. 7. Three main components of Gafgyt.

TABLE XVIII
COMMUNICATION FEATURES OF GAFGYT (PARTIAL)

| Feature | Description |
|---|---|
| **BUILD** | Compiled with the original Gafgyt source code and sends the architecture information of the infected device to the C&C server. |
| **Shelling** | Connecting to the port 666 of the C&C server and obtain the OS information of the IoT device. |
| **Katura** | Focusing on the expansion of payloads, up to 22 different vulnerability payloads are currently collected of *Gafgyt*. |
| **Boatnet** | Using as the online notification of the C&C servers in Russia. |
| **Arch** | A popular variant of BUILD. The hackers make a simple modification to counter the network protection rules. |

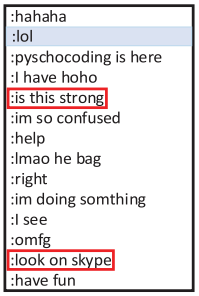



Fig. 8. Partial chat records in Gafgyt C&C Chatroom.

*Example 3 [Gafgyt (the Second Largest Family)]:* Gafgyt (also known as BASHLITE) is a kind of malware, which is written in C and designed to easily cross-compile to multiple computer architectures. It accounts for almost 27% of our data set. The source code of our Gafgyt samples consists of three parts: 1) downloader; 2) scanner; and 3) DDoS module. Fig. 7 shows an example of a Gafgyt sample.[5] Gafgyt creates a communication channel with the remote C&C server. The partially malicious features appearing in communication traffic that can be used to identify *Gafgyt* are shown in Table XVIII.

Meanwhile, we observe that Gafgyt has obvious Botnet as a Service (BaaS) characteristics. We trace that in the Chatroom of Gafgyt C&C, there are attackers dedicated to answering questions about the use of botnet service (Partial chat records are shown in Fig. 8).

Thus, Gafgyt has realized the transition from a traditional profit-taking model for selling attack traffic to providing botnet hosting services through the cloud platform. With such sales style, attack organization costs are reduced, and its attacks are

[5]MD5:845176498938d58f6653dec1984ba919.

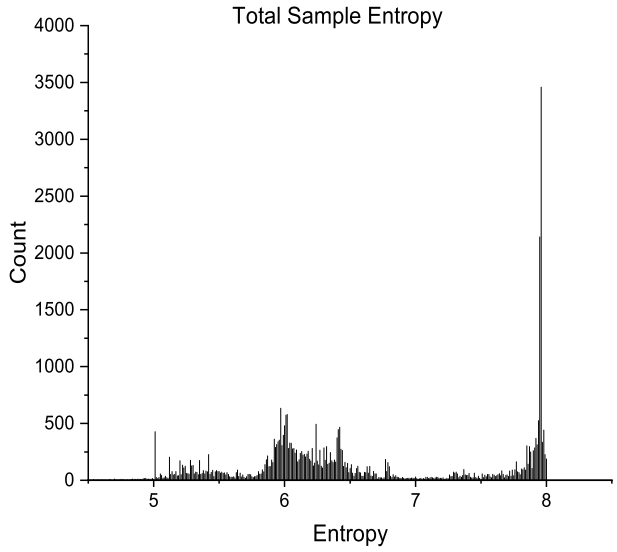



Fig. 9. Entropy distribution over the data set.

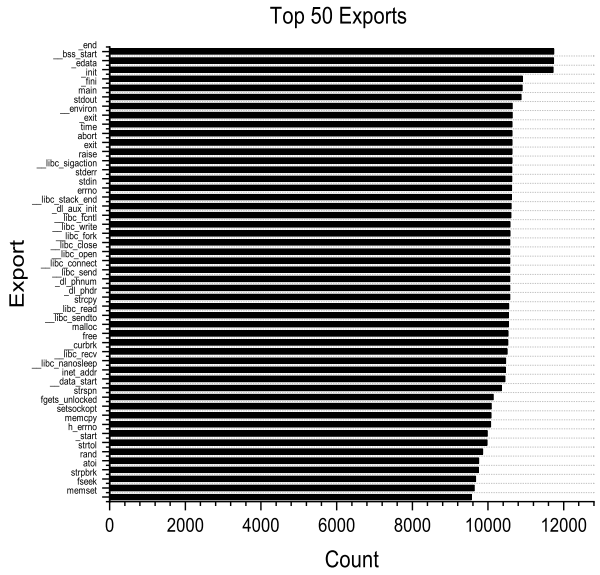



Fig. 11. Top 50 exports distribution over the data set.

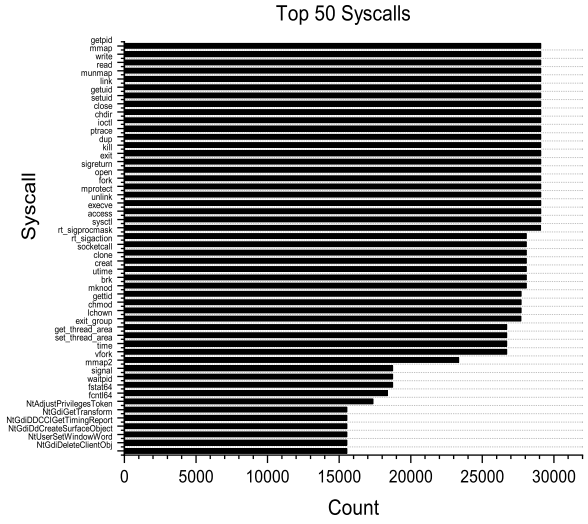



Fig. 10. Top 50 system calls distribution over the data set.

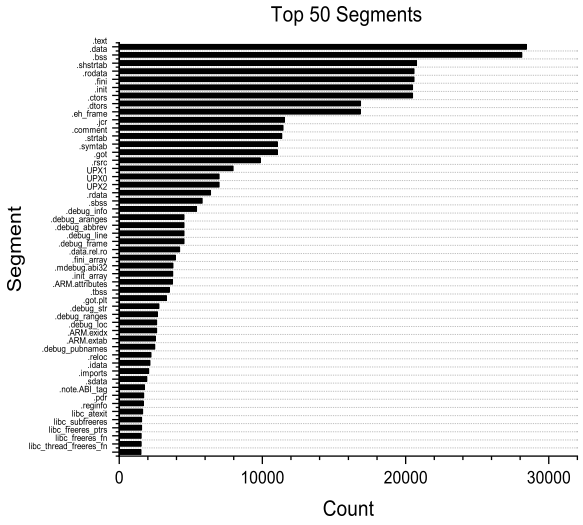



Fig. 12. Top 50 segments distribution over the data set.

bound to spread to various areas. As a consequence, the threat of IoT botnet enters a high level of danger.

## VIII. Trend Analysis and Limitation

We conducted numerous reverse analyses on the collected samples. The analytical results show that the number of Mirai variants (such as MIORI and SORA) increases. Based on our observation, the variants first execute the command "/bin/busybox SORA" to determine whether the victim has been infected. The variants then decide the next action based on the system feedback. This type of cautious behavior can improve the attack efficiency.

The traffic analysis by honeypots demonstrates that the proportion of IoT platform family attacks has further expanded. Gafgyt and Mirai contributed most of the attack behaviors. Overall, no significant change was observed in the DDoS characteristics, such as attack targets and C&C distribution.

From June 2019, we discovered an increasing number of fileless attacks for mining. These attacks are launching by executing the following commands "ifconfig; cat / proc / cpuinfo; ps|grep '[Mm] iner'; echo Hi|cat -n" to test the resource situation of the infected device. A good deal of traffic utilizes our honeypot as a redirector to conduct DDoS attacks.

Based on the analysis of the whole of 2019, we predict that the malware-based attacks of IoT will provide great attention to the smart devices, such as intelligent speaker, smart wearable devices, and edge computing device, in the next two years with the development of 5G. The IoT attacks will gradually become efficient, economical, and intelligent.

Our limitations are twofold: first, the literature-based study generally relies on authenticity and how detailed the attack has been described on the existing reports. Specifically, the public unknown behaviors and threats will never be mined by NLP. Hence, we present the sample analysis endeavor, which can offer feedback to the NLP result for verification and supplement. This situation brings us to our second limitation, which

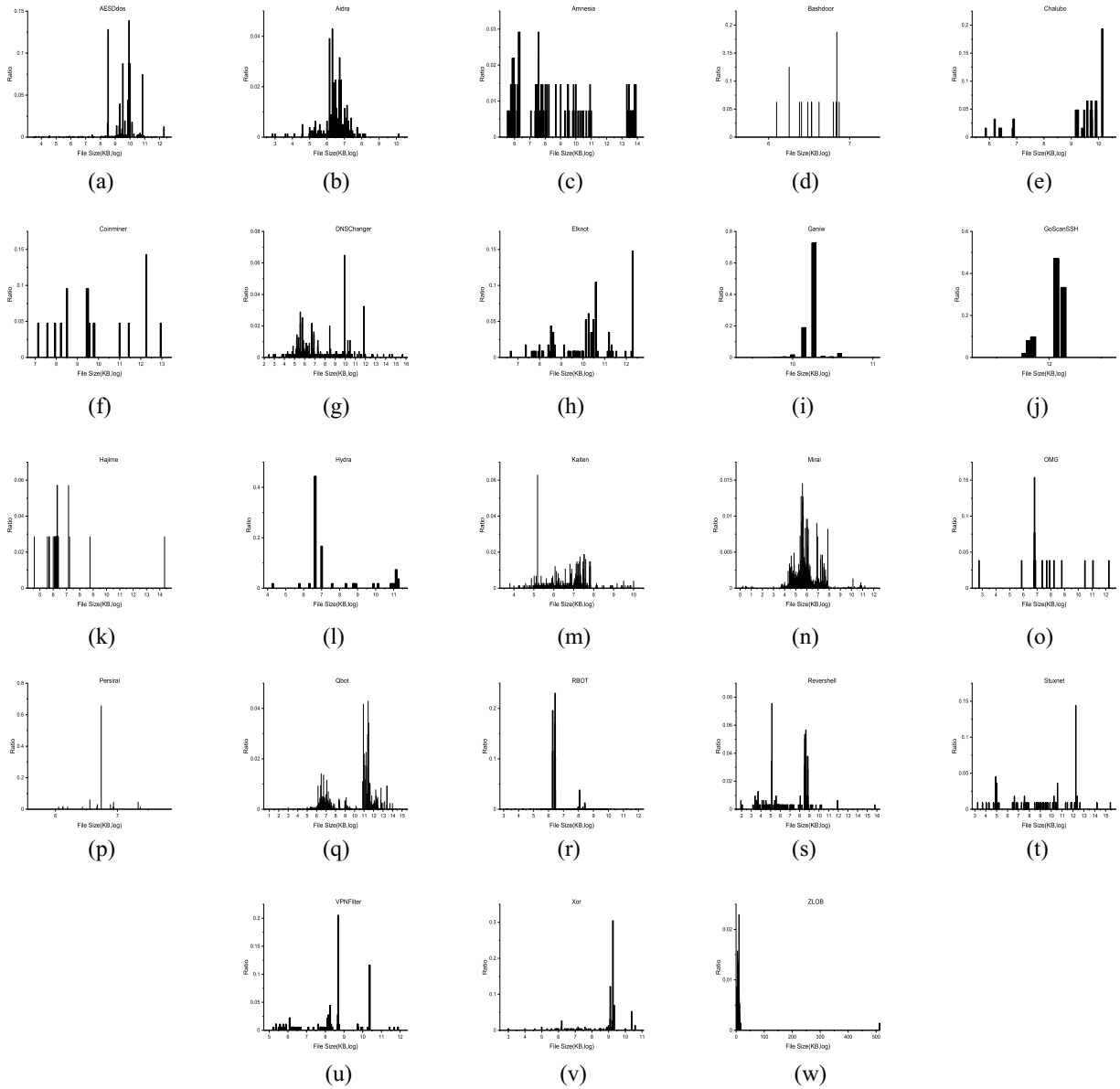



Fig. 13. File size distribution of ELF malware in the data set. (a) File size of AESDDoS. (b) File size of Aidra. (c) File size of amnesia. (d) File size of bashdoor. (e) File size of Chalubo. (f) File size of Coinminer. (g) File size of DNSChanger. (h) File size of Elknot. (i) Data unit Ganiw. (j) File size of GoScanSSH. (k) File size of Hajime. (l) File size of hydra. (m) File size of Kaiten. (n) File size of Mirai. (o) File size of OMG. (p) File size of Persirai. (q) File size of Qbot. (r) File size of RBOT. (s) File size of Revershell. (t) File size of Stuxnet. (u) File size of VPNFilter. (v) File size of Xor. (w) File size of ZLOB.

is also an open issue in IoT security. The data set of IoT malware is hard to collect (Section III). Specifically, the demand for a high-quality data set is endless.

## IX. Conclusion

IoT malware has become a rapidly growing threat. In this study, we proposed a comprehensive evolutionary study of IoT malware. Our work is twofold. We first presented a literature-based method to retrieve the existing malware behaviors and lineages among IoT malware families. Subsequently, the obtained behaviors were mapped to extractive features. Second, we designed an ensemble model for malware classification and lineage analysis. We evaluated our model with a large scale data set, including 38 963 malware covering 36 families and 16 752 benign binaries. We released all the scripts and samples to the security community for further study by other researchers. The feature set was optimized in the first step with the guide of the second step's analytical result. Besides, we verified the obtained 12 lineage relationships and discover five new lineages of IoT malware families with the help of the proposed model. Model performance can be improved by more behavioral features like network communication characteristics. Furthermore, we will optimize the NLP strategy to integrate more information from academic papers to improve prior knowledge integrity. Besides, we will adopt automated training technologies to save the model training cost, such as the architectural search functionality of AutoML.

## APPENDIX

See Figs. 7–13.

## REFERENCES


[1] D. Minoli, K. Sohraby, and B. Occhiogrosso, "IoT considerations, requirements, and architectures for smart buildings-energy optimization and next-generation building management systems," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 269–283, Feb. 2017.

[2] J. Chen *et al.*, "Your IoTs are (not) mine: On the remote binding between IoT devices and users," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, 2019, pp. 222–233.

[3] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[4] S. Ramanathan, J. Mirkovic, M. Yu, and Y. Zhang, "SENSS against volumetric DDoS attacks," in *Proc. 34th Annu. Comput. Security Appl. Conf. (ACSAC)*, San Juan, PR, USA, Dec. 2018, pp. 266–277. [Online]. Available: https://doi.org/10.1145/3274694.3274717

[5] I. U. Haq, S. Chica, J. Caballero, and S. Jha, "Malware lineage in the wild," *Comput. Security*, vol. 78, pp. 347–363, Sep. 2018.

[6] *Cisco Talos Intelligence Group—Comprehensive Threat Intelligence: CRAT Wants to Plunder Your Endpoints*. Accessed: 2021. [Online]. Available: https://blog.talosintelligence.com/2020/11/crat-and-plugins.html

[7] B. Botezatu. *New Hide—N Seek IoT Botnet Using Custom-Built Peer-to-Peer Communication Spotted in the Wild—Bitdefender Labs*. Accessed: 2021. [Online]. Available: https://labs.bitdefender.com/2018/01/new-hide-n-seek-iot-botnet-using-custom-built-peer-to-peer-communication-spotted-in-the-wild/

[8] *Home & Small Office Wireless Routers Exploited to Attack Gaming Servers*. Accessed: Oct. 2019. [Online]. Available: https://unit42.paloaltonetworks.com/home-small-office-wireless-routers-exploited-to-attack-gaming-servers/

[9] J. Su, V. D. Vasconcellos, S. Prasad, S. Daniele, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, 2018, pp. 664–669.

[10] A. Wang, R. Liang, X. Liu, Y. Zhang, K. Chen, and J. Li, "An inside look at IoT malware," in *Proc. Int. Conf. Ind IoT Technol. Appl.*, 2017, pp. 176–186.

[11] H. Griffioen and C. Doerr, "Examining Mirai's battle over the Internet of Things," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2020, pp. 743–756.

[12] H.-S. Ham, H.-H. Kim, M.-S. Kim, and M.-J. Choi, "Linear SVM-based Android malware detection," in *Proc. Front. Innov. Future Comput. Commun.*, 2014, pp. 575–585.

[13] W. Zhang, H. Wang, H. He, and P. Liu, "DAMBA: Detecting Android malware by ORGB analysis," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 55–69, Mar. 2020.

[14] H. Wang, H. He, and W. Zhang, "Demadroid: Object reference graph-based malware detection in Android," *Security Commun. Netw.*, vol. 2018, May 2018, Art. no. 7064131.

[15] S. D. Sl and C. Jaidhar, "Windows malware detector using convolutional neural network based on visualization images," *IEEE Trans. Emerg. Topics Comput.*, early access, Apr. 11, 2019, doi: 10.1109/TETC.2019.2910086.

[16] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding Linux malware," in *Proc. IEEE Symp. Security Privacy (SP)*, 2018, pp. 161–175.

[17] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bilge, and D. Balzarotti, "The tangled genealogy of IoT malware," in *Proc. Annu. Comput. Security Appl. Conf.*, Dec. 2020, pp. 1–16. [Online]. Available: https://dl.acm.org/doi/10.1145/3427228.3427256

[18] A. Calleja, J. Tapiador, and J. Caballero, "A look into 30 years of malware development from a software metrics perspective," in *Proc. Int. Symp. Res. Attacks Intrusions Defenses*, 2016, pp. 325–345.

[19] A. Calleja, J. Tapiador, and J. Caballero, "The malsource dataset: Quantifying complexity and code reuse in malware development," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 12, pp. 3175–3190, Dec. 2019.

[20] L. H. Park, J. Yu, H.-K. Kang, T. Lee, and T. Kwon, "Birds of a feature: Intrafamily clustering for version identification of packed malware," *IEEE Syst. J.*, vol. 14, no. 3, pp. 4545–4556, Sep. 2020.

[21] R. Langner, "StuxNet: Dissecting a cyberwarfare weapon," *IEEE Security Privacy*, vol. 9, no. 3, pp. 49–51, May/Jun. 2011.

[22] L. Durfina, J. Kroustek, and P. Zemek, "Psybot malware: A step-by-step decompilation case study," in *Proc. 20th Working Conf. Reverse Eng. (WCRE)*, 2013, pp. 449–456.

[23] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "DDoS-capable IoT malwares: Comparative analysis and mirai investigation," *Security Commun. Netw.*, vol. 2018, Feb. 2018, Art. no. 7178164.

[24] A. Costin and J. Zaddach, "IoT malware: Comprehensive survey, analysis framework and case studies," in *Proc. BlackHat USA*, 2018, pp. 33–39.

[25] E. Kovacs. (2015). *Large DDoS Botnet Powered by Routers Infected With Spike Malware*. [Online]. Available: https://www.securityweek.com/large-ddos-botnet-powered-routers-infected-spike-malware

[26] I. Arghire. (2019). *Aesddos Botnet Targets Vulnerability in Atlassian's Confluence Server*. [Online]. Available: https://www.securityweek.com/aesddos-botnet-targets-vulnerability-atlassian's-confluence-server

[27] Mcafee. *Mcafee Labs Threats Report Explores Threat Intelligence Sharing and Mirai Botnet*. Accessed: 2021. [Online]. Available: https://cyware.com/news/mcafee-labs-threats-report-explores-threat-intelligence-sharing

[28] ANTIY. (2018). *IoT Botnet Gafgyt and Netcore Backdoor*. [Online]. Available: https://www.antiy.com/response/Gafgyt&NetCore/IoT-Botnet-Gafgyt-and-NetCore-53413-Backdoor.pdf

[29] M. Smii. (2018). *Jenx, New IoT Botnet*. [Online]. Available: https://medium.com/secjuice/jenx-new-iot-botnet-c412d5a446ee

[30] I. Arghire. (2018). *Doubledoor: IoT Botnet Uses Two Backdoor Exploits*. [Online]. Available: https://www.securityweek.com/doubledoor-iot-botnet-uses-two-backdoor-exploits

[31] N. Security. (2018). *Doubledoor: IoT Botnet Bypasses Firewall as Well as Modem Security Using Two Backdoor Exploits*. [Online]. Available: https://blog.newskysecurity.com/doubledoor-iot-botnet-bypasses-firewall-as-well-as-modem-security-using-two-backdoor-exploits-88457627306d

[32] T. Spring. (2018). *Goscanssh Malware Targets SSH Servers, But Avoids Military and .Gov Systems*. [Online]. Available: https://threatpost.com/goscanssh-malware-targets-ssh-servers-but-avoids-military-and-gov-systems/130812/

[33] TI Team. (2018). *Seven New Mirai Variants and the Aspiring Cybercriminal Behind Them*. [Online]. Available: https://blog.avast.com/hacker-creates-seven-new-variants-of-the-mirai-botnet

[34] I. Group. (2013). *Carna Botnet—An Interesting, Amoral and Illegal Internet Census*. [Online]. Available: https://www.infosecurity-magazine.com/news/carna-botnet-an-interesting-amoral-and-illegal/

[35] Blaze. (2015). *Notes on Linux/XoR.DDoS*. [Online]. Available: https://bartblaze.blogspot.com/2015/09/notes-on-linuxxorddos.html

[36] W. L. Security. (2016). *Linux/Moose: Still Breathing*. [Online]. Available: https://www.welivesecurity.com/2016/11/02/linuxmoose-still-breathing/

[37] M. Beltov. (2017). *Amnesia IoT Botnet Infects Devices Worldwide*. [Online]. Available: https://bestsecuritysearch.com/amnesia-iot-botnet-infects-devices-worldwide/

[38] P. Paganini. (2016). *ELF Linux/Nyadrop, a New IoT Threat in the Wild*. [Online]. Available: https://securityaffairs.co/wordpress/52273/malware/elf-linux-nyadrop-iot.html

[39] N. Security. (2018). *Understanding the IoT Hacker—A Conversation With Owari/Sora IoT Botnet Author*. [Online]. Available: https://blog.newskysecurity.com/understanding-the-iot-hacker-a-conversation-with-owari-sora-iot-botnet-author-117feff56863

[40] J. Sanders. (2018). *Newly Discovered Slingshot Malware Was Hidden in Routers for 6 Years*. [Online]. Available: https://www.techrepublic.com/article/newly-discovered-slingshot-malware-was-hidden-in-routers-for-6-years/

[41] B. BOTEZATU. (2018). *Hide and Seek IoT Botnet Resurfaces With New Tricks, Persistence*. [Online]. Available: https://labs.bitdefender.com/2018/05/hide-and-seek-iot-botnet-resurfaces-with-new-tricks-persistence/

[42] Permalink. (2017). *Rickrolled by None Other Than IoTreaper*. [Online]. Available: https://labsblog.f-secure.com/2017/11/03/rickrolled-by-none-other-than-iotreaper/

[43] J. Biggs. (2017). *Brickerbot Is a Vigilante Worm That Destroys Insecure IoT Devices*. [Online]. Available: https://techcrunch.com/2017/04/25/brickerbot-is-a-vigilante-worm-that-destroys-insecure-iot-devices/

[44] J. Goldman. (2017). *Hajime Malware Infects Tens of Thousands of IoT Devices*. [Online]. Available: https://www.esecurityplanet.com/malware/hajime-malware-increasingly-targets-iot-devices.html

[45] Checkpoint. (2017). *IoTroop Botnet: The Full Investigation*. [Online]. Available: https://research.checkpoint.com/iotroop-botnet-full-investigation/

[46] D. Jain. (2017). *Technical Analysis Report on Rowdy, a New Type of IoT Malware Exploiting Stbs*. [Online]. Available: https://nsfocusglobal.com/technical-analysis-report-on-rowdy-a-new-type-of-iot-malware-exploiting/

[47] T. Micro. (2017). *Persirai: New Internet of Things (IoT) Botnet Targets IP Cameras*. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/persirai-new-internet-things-iot-botnet-targets-ip-cameras/

[48] Krebsonsecurity. (2018). *Alleged 'Satori' IoT Botnet Operator Sought Media Spotlight, Got Indicted*. [Online]. Available: https://krebsonsecurity.com/2018/09/alleged-satori-iot-botnet-operator-sought-media-spotlight-got-indicted/

[49] T. EASTON. (2018). *Chalubo Botnet Wants to DDoS From Your Server or IoT Device*. [Online]. Available: https://news.sophos.com/en-us/2018/10/22/chalubo-botnet-wants-to-ddos-from-your-server-or-iot-device/

[50] Cibersecurity. (2018). *Chalubo Botnet Wants to DDoS From Your Server or IoT Device*. [Online]. Available: https://www.cibersecurity.net.br/la-vem-mais-uma-botnet-a-cha-cha-lua/

[51] T. Spring. (2014). *Bashlite Family of Malware Infects 1 Million IoT Devices*. [Online]. Available: https://threatpost.com/bashlite-family-of-malware-infects-1-million-iot-devices/120230/

[52] S. Gatlan. (2014). *QBoT Malware Dropped Via Context-Aware Phishing Campaign*. [Online]. Available: https://www.bleepingcomputer.com/news/security/qbot-malware-dropped-via-context-aware-phishing-campaign/

[53] T. Spring. (2014). *Lizardstresser IoT Botnet Part of 400Gbps DDoS Attacks*. [Online]. Available: https://threatpost.com/lizardstresser-iot-botnet-part-of-400gbps-ddos-attacks/119006/

[54] S. News. (2016). *New Remaiten Malware Builds Botnet of Linux-Based Routers*. [Online]. Available: https://www.securityweek.com/new-remaiten-malware-builds-botnet-linux-based-routers

[55] Malwarebytes. (2007). *Trojan.dnschanger*. [Online]. Available: https://blog.malwarebytes.com/detections/trojan-dnschanger/

[56] Adaware. *A Closer Look at Zlob Trojans*. Accessed: 2021. [Online]. Available: https://www.adaware.com/knowledge-database/zlobtrojans

[57] ADAWARE. (2007). *What Are Exploit Kits?* [Online]. Available: https://www.cyber.nj.gov/threat-profiles/exploit-kits/

[58] F-secure. (2005). *Backdoor:w32/rbot*. [Online]. Available: https://www.f-secure.com/v-descs/rbot.shtml

[59] S. Networking. (2005). *Malware*. [Online]. Available: https://www.safer-networking.org/faq/malware/

[60] Microsoft. (2008). *Hacktool:Win32/Hydra*. [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=HackTool:Win32/Hydra

[61] A. Modine. (2010). *Chuck Norris Botnet Doesn't Infect Routers*. [Online]. Available: https://www.theregister.co.uk/2010/02/23/chuck_norris_botnet_doesnt_sleep/

[62] E. Tara. (2016). *Kaiten Malware Returns to Threaten IoT*. [Online]. Available: https://www.infosecurity-magazine.com/news/kaiten-malware-returns-to-threaten/

[63] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, "An Android application sandbox system for suspicious software detection," in *Proc. 5th Int. Conf. Malicious Unwanted Softw.*, 2010, pp. 55–62.

[64] X. Kou and Q. Wen, "Intrusion detection model based on Android," in *Proc. 4th IEEE Int. Conf. Broadband Netw. Multimedia Technol.*, 2011, pp. 624–628.

[65] W. Zhang, X. Li, N. Xiong, and A. V. Vasilakos, "Android platform-based individual privacy information protection system," *Pers. Ubiquitous Comput.*, vol. 20, no. 6, pp. 875–884, 2016.

[66] A.-D. Schmidt *et al.*, "Enhancing security of Linux-based Android devices," in *Proc. 15th Int. Linux Kongress*, 2008, pp. 1–16.

[67] L. Liu, G. Yan, X. Zhang, and S. Chen, "VirusMeter: Preventing your cellphone from spies," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, 2009, pp. 244–264.

[68] J. Cheng, S. H. Wong, H. Yang, and S. Lu, "SmartSiren: Virus detection and alert for smartphones," in *Proc. ACM 5th Int. Conf. Mobile Syst. Appl. Services*, 2007, pp. 258–271.

[69] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proc. 1st ACM Workshop Security Privacy Smartphones Mobile Devices*, 2011, pp. 15–26.

[70] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: A behavioral malware detection framework for Android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.

[71] B. Sun, X. Luo, M. Akiyama, T. Watanabe, and T. Mori, "PADetective: A systematic approach to automate detection of promotional attackers in mobile app store," *J. Inf. Process.*, vol. 26, pp. 212–223, 2018.

[72] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT," in *Proc. NDSS*, 2019, pp. 1–6.

[73] Y. Liu and A. Simpson, "AdSelector: A privacy-preserving advertisement selection mechanism for mobile devices," *Comput. J.*, vol. 60, no. 8, pp. 1251–1270, 2017.

[74] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "Scandroid: Automated security certification of Android," Comput. Sci. Dept., Univ. Maryland, College Park, MD, USA, Rep., 2009.

[75] W. Enck *et al.*, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, p. 5, 2014.

[76] Y. Liu and A. Simpson, "Privacy-preserving targeted mobile advertising: Requirements, design and a prototype implementation," *Softw. Practice Exp.*, vol. 46, no. 12, pp. 1657–1684, 2016.

[77] H.-S. Ham, H.-H. Kim, M.-S. Kim, and M.-J. Choi, "Linear SVM-based Android malware detection for reliable IoT services," *J. Appl. Math.*, vol. 2014, Sep. 2014, Art. no. 594501.

[78] *Trojan: Android/Zsone.A Description—F-Secure Labs*. Accessed: 2021. [Online]. Available: https://www.f-secure.com/v-descs/trojan_android_zsone.shtml

[79] B. Ruttenberg *et al.*, "Identifying shared software components to support malware forensics," in *Proc. Int. Conf. Detection Intrusions Malware Vulnerability Assessment*, 2014, pp. 21–40.

[80] M. Lindorfer, A. Di Federico, F. Maggi, P. M. Comparetti, and S. Zanero, "Lines of malicious code: Insights into the malicious software industry," in *Proc. ACM 28th Annu. Comput. Security Appl. Conf.*, 2012, pp. 349–358.

[81] D. R. Swanson, "Fish oil, Raynaud's syndrome, and undiscovered public knowledge," *Perspectives Biol. Med.*, vol. 30, no. 1, pp. 7–18, 1986.

[82] L. J. Jensen, J. Saric, and P. Bork, "Literature mining for the biologist: From information retrieval to biological discovery," *Nat. Rev. Genet.*, vol. 7, no. 2, p. 119, 2006.

[83] S. Spangler *et al.*, "Automated hypothesis generation based on mining scientific literature," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2014, pp. 1877–1886.

[84] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards automating risk assessment of mobile applications," in presented at the 22nd USENIX Security Symp. (USENIX Security), 2013, pp. 527–542.

[85] Z. Zhu and T. Dumitraş, "FeatureSmith: Automatically engineering features for malware detection by mining the security literature," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 767–778.

[86] Matrix. *Tactics Listed by Wenterprise Matrix*. Accessed: 2021. [Online]. Available: https://attack.mitre.org/matrices/enterprise/

[87] F. Bussoletti. *QAKBOT (AKA QBOT) Malware Is Evolving With New Obfuscation Techniques*. Accessed: 2021. [Online]. Available: https://www.difesaesicurezza.com/en/cyber-en/qakbot-aka-qbot-malware-is-evolving-with-new-obfuscation-techniques/

[88] M.-C. De Marneffe and C. D. Manning, "Stanford typed dependencies manual," Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Rep., 2008.

[89] G. Author. *Inside the Infamous Mirai IoT Botnet: A Retrospective Analysis*. Accessed: 2021. [Online]. Available: https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/#toc-1

[90] Princeton University. *WordNet*. Accessed: 2021. [Online]. Available: https://wordnet.princeton.edu/

[91] B. Liu, M. Hu, and J. Cheng, "Opinion observer: Analyzing and comparing opinions on the Web," in *Proc. 14th Int. Conf. World Wide Web*, 2005, pp. 342–351.

[92] M. Hu and B. Liu, "Mining opinion features in customer reviews," in *Proc. AAAI*, vol. 4, 2004, pp. 755–760.

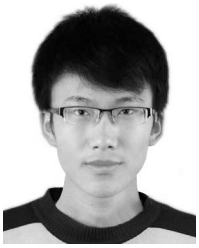
[93] M. Kerrisk. *Linux Man-Pages Project*. Accessed: 2021. [Online]. Available: kernel.org/doc/man-pages/

[94] Python. *Nltk*. Accessed: 2021. [Online]. Available: http://www.nltk.org/

[95] M. Malik. *Meet Remaiten—A Linux Bot on Steroids Targeting Routers and Potentially Other IoT Devices*. Accessed: 2021. [Online]. Available: https://www.welivesecurity.com/2016/03/30/

[96] Cowrie. *Cowrie Project*. Accessed: 2021. [Online]. Available: https://github.com/cowrie/cowrie

[97] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVCLASS: A tool for massive malware labeling," in *Proc. Int. Symp. Res. Attacks Intrusions Defenses*, 2016, pp. 230–253.
[98] V. Iyengar. (2018). *Project of Sandbox, Detux*. [Online]. Available: https://github.com/detuxsandbox/detux
[99] Radare. (2018). *Access Radare2 Via Pipe From Any Programming Language*. [Online]. Available: https://github.com/radare/radare2-r2pipe
[100] Y. Zhang, Q. Huang, X. Ma, Z. Yang, and J. Jiang, "Using multi-features and ensemble learning method for imbalanced malware classification," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Tianjin, China, 2016, pp. 965–973.
[101] Y. Liu and X. Yao, "Ensemble learning via negative correlation," *Neural Netw.*, vol. 12, no. 10, pp. 1399–1404, 1999.
[102] Python. *GridSearchCV*. Accessed: 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[103] C. D. Manning, H. Schütze, and P. Raghavan, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.



**Huanran Wang** received the B.S. degree in software engineering from Harbin Engineering University, Harbin, China, in 2016. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin.

His research interests include mobile security, secure Internet of Things, and cyberspace security.

**Weizhe Zhang** (Senior Member, IEEE) received the B.Eng., M.Eng., and Ph.D. degrees in computer science and technology from Harbin Institute of Technology, Harbin, China, in 1999, 2001, and 2006, respectively.

He is currently a Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, and the Director of the Cyberspace Security Research Center, Pengcheng Laboratory, Shenzhen, China. He conducts research in high-performance computing, parallel and distributed system, cloud computing, real-time computing, and computer network and security. He has published more than 100 academic papers in journals, books, and conference proceedings.

**Hui He** (Member, IEEE) received the Ph.D. degree from the Department of Computer Science, Harbin Institute of Technology, Harbin, China, in 2006.

She is currently a Full Professor with the Network Security Center, Department of Computer Science, Harbin Institute of Technology. Her research interests are mainly focused on distributed computing, IoT, and big data analysis.

**Peng Liu** (Member, IEEE) received the Ph.D. degree from George Mason University, Fairfax, VA, USA, in 1999.

He is the Raymond G. Tronzo, MD Professor of Cybersecurity, and the Founding Director of the Center for Cyber-Security, Information Privacy, and Trust, Penn State University, University Park, PA, USA. His research interests are in all areas of computer security. He has published over 300 technical papers.

**Daniel Xiapu Luo** received the Ph.D. degree in computer science from The Hong Kong Polytechnic University, Hong Kong, in 2007.

He is an Associate Professor with the Department of Computing, The Hong Kong Polytechnic University. His current research interests include mobile/IoT security and privacy, blockchain, network security and privacy, and software engineering. His work appeared in top venues.

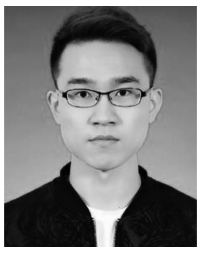
Dr. Luo has received seven best paper awards, such as INFOCOM'18, ISPEC'17, and ISSRE'16.

**Yang Liu** received the B.Eng. degree in computer science from Ocean University of China, Qingdao, China, the M.Sc. degree in software engineering from Peking University, Beijing, China, and the Ph.D. degree in computer science from University of Oxford, Oxford, U.K., in July 2018, under the guidance of Prof. A. Simpson.

He is currently an Assistant Professor with the Department of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He is interested in security and privacy problems and, in particular, the privacy issues related to mobile and IoT devices.

**Jiawei Jiang** received the B.S. degree in information security from Harbin Institute of Technology, Harbin, China, in 2019, where he is currently pursuing the master's degree with the School of Cyber Security.

His research interests include mobile security, secure Internet of Things, and cyberspace security.



**Yan Li** received the B.S. degree in information security from Harbin Institute of Technology, Harbin, China, in 2019, where he is currently pursuing the M.S. degree in cyberspace security.

His research interests include edge computing, cloud computing, and secure Internet of Things.

**Xing Zhang** received the Ph.D. degree in information security from Peking University, Beijing, China, in 2016.

He is a Senior Security Researcher with NSFOCUS, Beijing. During the first two years in NSFOCUS, he was also working with Tsinghua University, Beijing, as a Postdoctoral Fellow. He is focusing on IoT security.

**Wenmao Liu** received the Ph.D. degree in information security from Harbin Institute of Technology, Harbin, China, in 2013.

He served as a Researcher with NSFOCUS, Inc., Beijing, China. He is the Director of Innovation Center, and also the Leader of XingYun Lab, NSFOCUS Inc., the Co-Chair of Cloud Security Service WG, CSA. During the first two years in NSFOCUS, he was also working with Tsinghua University, Beijing, as a Postdoctoral Fellow. His interests are focused on cloud security, IoT security, data-driven analytics, and other new research areas of network security. Now, he has been promoting the adoption of cloud native security, edge computing, and 5G security.

Dr. Liu works closely with academia, he is a member of a China Computer Federation council.

**Runzi Zhang** (Member, IEEE) was born in Shandong, China, in 1989. He received the Ph.D. degree from the National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing, China, and the University of Chinese Academy of Sciences, Beijing.

He is a Postdoctoral Fellow with Tsinghua University, Beijing. He is also a Security Researcher with NSFOCUS Technologies Group, Beijing. His research interests include data-driven security operations and threat hunting.

**Xing Lan**, photograph and biography not available at the time of publication.