Understanding and Securing Device Vulnerabilities through Automated Bug Report Analysis

Xuan Feng[1,2,5]*, Xiaojing Liao[2], XiaoFeng Wang[2], Haining Wang[3]
Qiang Li[4], Kai Yang[1,5], Hongsong Zhu[1,5], Limin Sun[1,5]†

[1] *Beijing Key Laboratory of IoT Information Security Technology, IIE‡, CAS, China*
[2] *Department of Computer Science, Indiana University Bloomington, USA*
[3] *Department of Electrical and Computer Engineering, University of Delaware, USA*
[4] *School of Computer and Information Technology, Beijing Jiaotong University, China*
[5] *School of Cyber Security, University of Chinese Academy of Sciences , China*

Abstract

Recent years have witnessed the rise of Internet-of-Things (IoT) based cyber attacks. These attacks, as expected, are launched from compromised IoT devices by exploiting security flaws already known. Less clear, however, are the fundamental causes of the pervasiveness of IoT device vulnerabilities and their security implications, particularly in how they affect ongoing cybercrimes. To better understand the problems and seek effective means to suppress the wave of IoT-based attacks, we conduct a comprehensive study based on a large number of real-world attack traces collected from our honeypots, attack tools purchased from the underground, and information collected from high-profile IoT attacks. This study sheds new light on the device vulnerabilities of today's IoT systems and their security implications: ongoing cyber attacks heavily rely on these known vulnerabilities and the attack code released through their reports; on the other hand, such a reliance on known vulnerabilities can actually be used against adversaries. The same bug reports that enable the development of an attack at an exceedingly low cost can also be leveraged to extract vulnerability-specific features that help stop the attack. In particular, we leverage Natural Language Processing (NLP) to automatically collect and analyze more than 7,500 security reports (with 12,286 security critical IoT flaws in total) scattered across bug-reporting blogs, forums, and mailing lists on the Internet. We show that signatures can be automatically generated through an NLP-based report analysis, and be used by intrusion detection or firewall systems to effectively mitigate the threats from today's IoT-based attacks.

1 Introduction

The pervasiveness of Internet-of-Things (IoT) systems, ranging from cameras, routers, and printers to various home automation, industrial control and medical systems, also brings in new security challenges: they are more vulnerable than conventional computing systems. IoT systems may fall into an adversary's grip once their vulnerabilities are exposed. Indeed, recent years have seen a wave of high-profile IoT related cyber attacks, with prominent examples including IoT Reaper [25], Hajime [24], and Mirai [41]. Particularly, compromised IoT systems are becoming the mainstay for today's botnets, playing a central role in recent distributed denial-of-service attacks and other cybercrimes [39]. An example is Mirai, which involved hundreds of thousands of IoT devices and generated an attack volume at 600 Gbps, overwhelming Krebs on Security, OVH, and Dyn. It does not come as a surprise that these devices are reported to be hacked all through known vulnerabilities, including those caused by misconfiguration or mismanagement, just like unpatched servers and personal computers that are routinely exploited [42]. Still less clear, however, are the fundamental causes for this trend of malicious activities that are disproportionately related to IoT systems and these vulnerable devices' impact upon the cybercrime ecosystem.

Understanding the perilous IoT world. More specifically, we raise the following questions: Why are IoT devices more favorable to cybercriminals than other Internet hosts? How important are known vulnerabilities to the ongoing attacks on IoT systems? Is there an effective defense to mitigate ongoing attacks? The answers to these questions are critical for seeking an effective means to thwart the wave of malicious attacks that increasingly rely on a large number of vulnerable IoT devices. Finding these answers, however, is by no means trivial due to the challenges in (1) recovering disclosed IoT vulnerabilities from a large number of vulnerability reports scattered around forums, mailing lists, and blogs, and (2) collecting artifacts from the cybercrime underground to study how known flaws are utilized and how significant they are to ongoing criminal activities.

To understand how cybercriminals use these known vulnerabilities, we set up honeypots to collect the data of real-world IoT exploits and also analyzed four popular attack toolkits. From the adversary's end, our study shows that today's IoT

*Work was done while visiting Indiana University Bloomington.
†Corresponding author
‡Institute of Information Engineering

attacks almost *exclusively* use known vulnerabilities for exploitation. Specifically, among 81 different exploit scripts recovered from our honeypots, we found that 78 of them are on our vulnerability list. Also, each of the four attack toolkits we studied incorporates the exploits on at least 34 vulnerabilities, with all of them on our list. More importantly, we evidence that an adversary extensively leverages the exploit code released together with the vulnerability reports, and in most cases, directly copies the code or slightly adjusts it. More than 80% of the IoT-related reports come with working attack methods. Given our observations that *most of* IoT vulnerabilities can be exploited to attack target devices (compared with only 5% exploitable ones in Linux kernel vulnerabilities [43]), it is apparent that the cost for attacking IoT systems today is *exceedingly low*.

Automated protection generation. On the other hand, we show that adversaries' indulgence in such low-hanging fruits can actually be used against themselves. Specifically, the reliance on known vulnerabilities means that a large attack surface would be closed once these problems have been fixed. Interestingly, this turns out to be completely feasible, due to the simplicity of the problems and the availability of the vulnerability disclosure and attack code that carries all the information we need to fix the reported flaws. Based on this observation, we developed a framework, called *IoTShield*, which utilizes Natural-Language Processing (NLP) to automatically evaluate the content of vulnerability reports. In particular, IoTShield is based upon a set of automatic content analysis techniques that accurately discover IoT-related vulnerability disclosures from a large number of vulnerability reports published at different sources.

Using the approaches above, we automatically analyzed 430,000 vulnerability reports and discovered more than 7,500 IoT reports in the past 20 years. Then, IoTShield extracts key knowledge from these reports to automatically generate *vulnerability-specific* signatures, which can be used by existing intrusion detection systems (IDSes) or web application firewalls (WAFs) to screen the traffic received by IoT devices under protection. We validated the efficacy of IoTShield over 178,778 traces harvested by our honeypots, as well as 11,602 traces of eight real devices, including both attack and legitimate traffic. Furthermore, we evaluated the effectiveness of IoTShield over a long-time (more than one year) traffic captured in an industrial control system's human machine interface (HMI) Honeypot. The evaluation results show that IoTShield achieves a high precision (above 97%) and a very low false positive rate with minor performance impact.

Contributions. The major contributions of this work are outlined as follows:

• *New discovery*. Leveraging the collected traces of real-world IoT exploits and analysis results from popular attack toolkits, we bring to light new observations, including the adversary's exclusive use of known flaws and published code. These observations demonstrate that IoT devices are indeed more vulnerable, less patchable, and easier to attack than traditional Internet hosts, elucidating the ongoing cybercrime trend that heavily relies on these devices.

• *New defense*. More importantly, these findings also present us with an opportunity that could lead to the immediate defeat of most attack vectors in today's IoT-related attacks. We demonstrate that from the same sources adversaries use to build their exploits, vulnerability signatures can be *automatically* generated using NLP, and be quickly deployed to shield IoT devices from malicious attacks. Given the adversary's dependence on these known vulnerabilities, we believe that this simple, low cost yet effective defense will significantly raise the bar for future IoT-related attacks.

Roadmap. The rest of the paper is organized as follows. Section 2 presents the background and our threat model. Section 3 describes our new understandings on IoT vulnerabilities and real-world threats to IoT devices. Section 4 introduces the automated protection generation to defend against IoT attacks. Sections 5 and 6 present the implementation and evaluation of our automated protection generation, respectively. Section 7 discusses the limitations of this study and mitigation. Section 8 surveys related work, and finally, Section 9 concludes the paper.

2 Background

IoT vulnerability life cycle. Usually, an IoT device's vulnerability is a loophole in its firmware that enables an attacker to circumvent the deployed security measures [57]. Such a vulnerability has a life cycle with distinct phases characterized by its discovery, disclosure, exploitation, and patching. The first phase starts when the vulnerability is discovered by a vendor, a hacker, or a third-party security researcher. The security risk becomes particularly high if it is first found by hackers. The next phase is the public disclosure of the vulnerability by those who discover it, and is supposed to be done through a coordinated process [22], during which the vulnerability information is kept confidential allowing vendors to create a patch. However, this procedure is not always followed. Actual real-world disclosures could happen in different ways through sources across the Internet, including personal blogs, public forums, and security mailing lists. Once publicly disclosed, the information about a vulnerability is freely available to anyone. Thus, the level of security risk further increases as the hacker community is active in developing and releasing zero-day exploits [40]. From our observations, about 80% of IoT vulnerability reports are released together with exploitable methods, which can be readily utilized by hackers.

Even worse, a vendor may not provide any security updates or patches in response to a disclosure, even though it is supposed to do so. Also, even with the patches available, it is not uncommon that many users of the affected IoT devices do not

install them, given the complicated procedure (for ordinary users) to patch the firmware.

The life cycle of an IoT vulnerability ends when all IoT users install the patch to fix the vulnerability. However, even if an IoT device has a serious security vulnerability and vendors have released the patch, some users have no capability of updating patches in a timely manner due to their limited knowledge. From our observations, the life cycle of some IoT vulnerabilities lasts more than five years, during which these problems can be exploited at any time.

Signature-based IDS. An intrusion detection system (IDS) monitors a network or a system for malicious activities or policy violations. These systems can be signature-based or anomaly-based. Signature-based detection leverages known patterns (signatures) of malicious code or operations to identify malicious activities, while anomaly-based detection captures deviations from a system's normal profile. The focus of our protection is to provide automatic signature generation for the signature-based detection systems, such as Snort [37].

Signatures can be used to describe a specific attack on a vulnerability or model the vulnerability itself. The latter, which provides comprehensive protection against *all* related attacks on the weakness, is called a *vulnerability-specific* signature. Such a signature is typically created through manual analysis of a vulnerability. In our study, however, we found that the rich information about IoT vulnerabilities from various sources (blogs, mailing lists, and forums) can actually be leveraged to *automatically* generate such a signature, using NLP techniques.

Natural language processing. Our research utilizes various NLP techniques to analyze the text content of various vulnerability reports. A spell-checking [1] is used to filter out documents irrelevant to IoT. Regular expression based pattern matching is utilized to identify the vulnerability reports related to IoT. Further, semantic consistency analysis is used to examine the extracted IoT entities. To generate a vulnerability-specific signature, in-depth understanding of the vulnerability semantics is needed and can be achieved by using a grammatical dependency parser [51], which provides a representation of grammatical relations between words in a sentence.

Threat model. In this work, we consider an adversary who attempts to exploit the security flaws disclosed in a vulnerability report to compromise remote, Internet-connected IoT devices. For this purpose, the adversary can compromise the remote IoT devices and use them to launch malicious attacks. In particular, we focus on Internet-connected IoT devices (e.g., IP cameras, routers, and printers) that expose their attack surfaces on the Internet. Accordingly, adversaries can exploit those devices' security flaws remotely, and gain unauthorized control of those vulnerable devices (e.g., a compromised IoT device may become a botnet node).

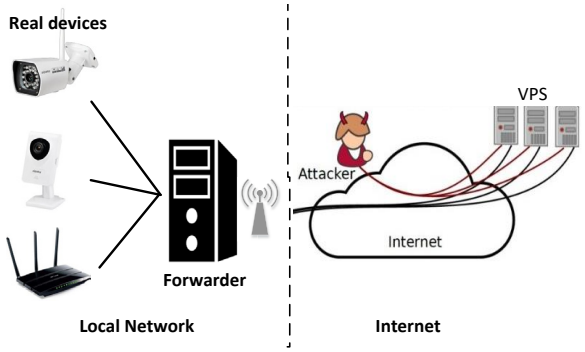

Figure 1: The infrastructure of our real device honeypots.

3 Understanding Real-World Threats

To understand how IoT vulnerabilities are exploited by adversaries, we deployed honeypots, analyzed underground attack toolkits, and studied prior attack reports. Here we elaborate our findings.

3.1 Honeypot

Our honeypots include real IoT devices and simulated devices for collecting real-world attack activities.

Real device honeypot. We deployed eight vulnerable IoT devices (three routers and five cameras) as honeypots from May 2018 to June 2018. These devices and their corresponding vulnerabilities are listed in Table 1. We chose these devices because they are typical IoT systems being exploited in various real-world attacks [41].

Figure 1 illustrates our honeypot system's infrastructure. To increase these devices' IP diversity, we rented Virtual Private Servers (VPSes) across different countries as relay hosts for each IoT device. Whenever an attacker connects to a VPS, we redirect this request to the corresponding IoT device and forward its response back to the attacker.

More specifically, we use the reverse SSH tunneling [33] to bridge the gap between the IoT devices behind NAT and VPSes. For this purpose, we set up a persistent SSH tunnel from our device to its corresponding VPS, which is configured with the SSH port forwarding command [38] to send the traffic received from the VPS' public IP on the HTTP port to FORWARD_PORT (pointing to the IoT device). For devices not supporting the SSH protocol, we set up a PC as FORWARD_PORT, passing the received traffic to the device.

Simulated honeypot. To study the attacks on more vulnerable devices, we also deployed four simulated IoT honeypots from May 2018 to July 2018[1] across two countries (Canada and the United States) and four cities (Buffalo, Los Angeles,

[1]The timeline for simulated honeypot data collection is not consistent with that of the real device honeypot, due to the data loss caused by a misconfiguration event in our real device honeypot.

Table 1: IoT device honeypot.

Products	Vulnerabilities
Linksys WVC54GCA	Multiple XSS [10]
	Absolute path traversal [12]
	Stored passwords/keys [13]
	Directory traversal (adm/file.cgi) [11]
TP-Link TL-SC3171G	Authentication bypass [19]
	OS Command injection [14]
	Hard-coded credentials [15]
	Unauthenticated file uploads [16]
	Unauthenticated firmware upgrades [17]
Dahua IPC-HF2100	Hard-coded Credentials [18]
D-Link DIR-645	Authentication Bypass [21]
TVT TD-9436T	Command execution [32]
Easyn Model:10D	Unreported 0-day
TP-link TL-WAR458L	Remote command execution [20]
TP-Link TL-WR941N	Backdoor [8]

Table 2: Four simulated honeypots.

Name	Vulnerability	Affected products
D-link SOAP [23]	command injection	at least 12 products
GoAhead webs [27]	7 CVEs	1,250 affected models
JAWS [30]	command injection	all DVR running JAWS
Avtech [4]	14 vulnerabilities	all Avtech devices

Table 3: Traffic analysis of deployed honeypots.

	Real devices	Simulated honeypots
Malicious (Targeted)	20	∼300
Malicious (Blind-scanned)	121	∼1,560
Benign	11,451	176,764
Unknown	10	∼154
Total	11,602	178,778

Canyon Country, and Beauharnois), which cover more than 2,000 devices and 23 vulnerabilities.

The settings of these simulators are listed in Table 2. The Avtech honeypot covers 14 vulnerabilities, such as authentication bypass and command injection, which affect all Avtech devices (i.e., IP camera, NVR, and DVR) and firmware versions. The "GoAhead webs" honeypot simulates the GoAhead IP camera using a GoAhead-webs HTTP server. The honeypot covers seven critical vulnerabilities, such as the backdoor account and the pre-auth information leakage. Particularly, the latter is found in more than 1,250 different camera models. This enables us to significantly increase the number of vulnerable devices in our study.

Specifically, each honeypot is an HTTP server (on Apache), whose default configurations (such as default page and HTTP response header/body) have been modified to simulate real devices. If a honeypot finds an IP address that attempts to connect to our honeypot, it records the request packets from the IP and their timestamps before sending back "200 OK" and redirecting all HTTP requests to our main page. Note that some attacks may first send a harmless request to identify their target devices. In this case, returning a "200 OK" often triggers follow-up attack behaviors. Also, all of our simulators are indexed as real IoT devices in Shodan [59], and so they can be discovered from the device search engine.

Analysis. From May to July in 2018, our honeypots gathered 190,380 HTTP requests from 47,089 IPs across 175 countries. We analyzed these traffic traces as follows: first we removed those confirmed to be legitimate, including legitimate login attempts, the requests like "`GET /`", and other ordinary HTTP GET requests; then we scanned the traces for attack attempts by searching the exploit code in our vulnerability dataset and further looking for the common attack traffic patterns, such as the presence of SQL commands and various execution commands (e.g., "`cmd=/usr/bin/telnetd`"). In this way, we identified nearly 2,000 IoT exploit attempts from 60 different countries, with an average of 38 attacks per day.

Table 3 presents the results in detail: for the real device honeypot, 141 unique attacks with 26 different scripts were captured, and 1,860 unique attacks through 81 attack scripts were found from our simulated honeypots. About 164 *unknown* requests still cannot be confirmed to be legitimate or malicious.

Analyzing these attacks (2,001 in total), we found that about 320 of them aimed at the honeypot (real or simulated) devices, while about 1,681 targeted the devices whose types are not covered in any honeypots. On one hand, this indicates that an adversary may blindly conduct an attack without first identifying the device type. On the other hand, this implies that our honeypots have a wide-spectrum attack coverage, not limited by the types of devices deployed at honeypots. More importantly, there are only 164 unknown requests observed by our honeypots. Even if we *conservatively* assume that all unknown requests originate from individual malicious attacks that exploit unknown security flaws, less than 10% (164 vs 2,001) of the total attacks exposed to our honeypots are such unknown attacks. In other words, more than 90% of malicious attacks exploit the *known* vulnerabilities. This observation is consistent with our analysis of underground attack toolkits (see Section 3.2).

Most commonly exploited vulnerabilities found in our study include unauthenticated command injection and information disclosure. These flaws can be easily attacked by sending a simple HTTP request to the vulnerable device to gain full control of the device. An intriguing observation is that 96% of the IoT attacks use the same or similar scripts included in the vulnerability reports we collected. For example, an attacker compromised our device TVT TD-9436T through a command execution vulnerability by using the exact same code documented in the report [32]. In another exploit (i.e., `/board.cgi?cmd=/usr/sbin/telnetd`) [28] on the same vulnerability location "board.cgi", the only change found in the attack code was an adjustment of a parameter from "`/usr/sbin/telnetd`" to the Linux command "`cat+/etc/passwd`".

Table 4: Underground IoT attack tools.

Name	Vulnerabilities
IPCAM exploits	Pre-Auth Info Leak
Huawei Exploits	Command Execution
iotNigger	Netis Backdoor
Brickerbot	More than 30 vulnerabilities

Table 5: Known IoT attack activities.

Name	Vulnerabilities	Year
IOT Reaper [25]	10 vulnerabilities	2017
Hajime [24]	at least 3 vulnerabilities	2016
Satori [34]	2 vulnerabilities	2018
Brickerbot [6]	21 vulnerabilities	2017
Masuta [26]	bypass & command execution	2018
Amnesia [3]	remote code execution	2017

3.2 Artifacts from Other Sources

To validate the findings made from the honeypots, we further analyzed four underground attack toolkits and six well-documented IoT botnets, which are elaborated below.

Underground attack tools. In this work, we searched popular underground marketplaces (such as openbazaar and dreammarket) by using a set of keywords related to attack toolkits (such as IoT malware names listed in Table 12 of Appendix), in an attempt to find the posts selling such tools. Once the posts were discovered, we contacted the sellers and purchased the tools. Altogether, we obtained four such tools with their source code (see Table 4), including Brickerbot, a variation for the one used in the famous Brickerbot attack [6].

By analyzing their code, we again found that *all* vulnerabilities they exploit are known ones, and their attack scripts are *all* copied from the vulnerability reports with minor changes (e.g., C&C server IP) to customize for specific attack campaigns. More specifically, from these tools, we identified 99 different attack scripts related to 34 vulnerabilities. Those vulnerabilities are all recorded in our dataset. Among the 99 attack scripts, three of them are exactly the same as those documented, while the remaining 96 all have small changes. As an example in Table 4, the Huawei exploit on an arbitrary command execution vulnerability apparently comes from exploits-db (https://www.exploit-db.com/exploits/43414/), with only its Linux command (e.g., "ls") changed to a new one (e.g., communicate with a specific remote server). Again, this confirms what we observed by analyzing our honeypot data: most IoT attacks utilize known vulnerabilities and even the attack code provided in the vulnerability reports, which on the other hand could be leveraged to quickly suppress this emerging attack wave.

Known attacks. Finally, we analyzed some well-known, well-documented IoT botnets that were recently reported (Table 5) to understand whether mostly known vulnerabilities and documented attack scripts were indeed used. These botnets were all aimed at the security flaws that affect many different products (like IoT Reaper) or those widely deployed on the Internet (like Masuta). Some of them (IoT Reaper and Amnesia) also focus on the vulnerabilities without patches.

Once again, we found that all the vulnerabilities exploited in these attacks are also included in the reports gathered in our research, and all the scripts attacking these flaws are the copies or variations of the code in the reports. For example, IoT Reaper attacks 37 vulnerabilities documented by 10 different reports: 10 for remote command execution, at least 24 not on any CVE and 7 without patches. Note that *all* these attacks took place *after* the disclosure of related vulnerabilities. For instance, IoT Reaper was brought to light in 2017, while the flaw it uses, Linksys E1500/E2500 vulnerabilities, was made public in 2013.

4 Automated Vulnerability-specific Signature Generation

As mentioned earlier, the IoT vulnerability ecosystem has a serious problem: the attack scripts are often publicly available in the vulnerability reports, making those known vulnerabilities easy to exploit. Such a problem has led to the spree of large-scale IoT based attacks in recent years, which almost exclusively exploit known flaws.

In the meantime, the adversary's reliance on the well-documented vulnerabilities also presents a new opportunity for mitigating such threats. From the same vulnerability reports, vulnerability-specific information can be extracted to form a protection strategy that stops all attacks on the vulnerability. In our research, we developed IoTShield, an automatic tool that collects IoT vulnerability reports from the Internet, analyzes the content of the IoT vulnerability reports, and recovers key knowledge to generate *vulnerability-specific* signatures, with their qualities determined by the comprehensiveness of the reports. These signatures can be easily deployed to existing intrusion detection systems (IDSes) or web application firewalls (WAFs) to detect exploit attempts on the target device from the traffic it receives. In the following, we elaborate on this approach.

4.1 Overview and Data

Collecting IoT vulnerability reports from Internet and extracting vulnerability-specific knowledge from the collected IoT vulnerability reports for signature generation is a non-trivial task, with unique technical challenges. First, a large number of vulnerability reports are scattered around forums, mailing lists, and blogs with different format written by different people. It is difficult to identify IoT vulnerability reports from other documents. Second, such identified IoT vulnerability reports describe security flaws in natural language, which makes a large-scale discovery of vulnerability information difficult.

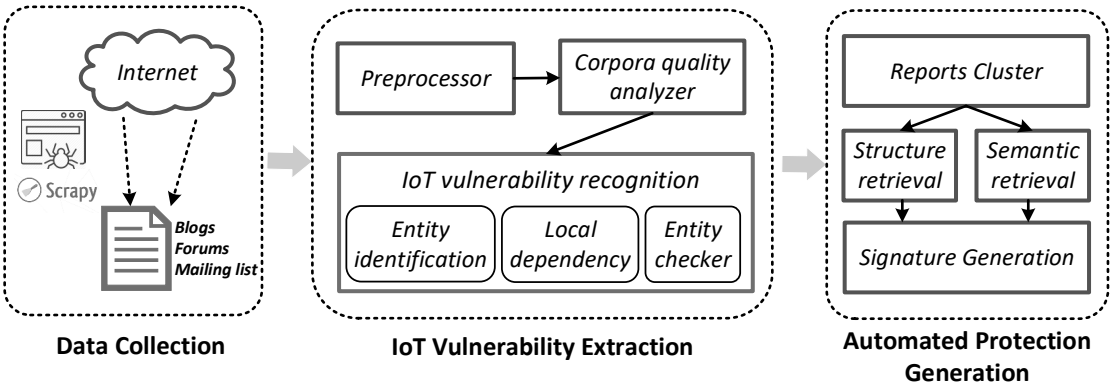

Figure 2: The architecture of IoTShield.

Third, identifying critical elements for a signature requires domain-specific knowledge to carefully distinguish between exploit-specific information and vulnerability-specific information. To address these challenges, in IoTShield, we use an IoT vulnerability extractor to remove irrelevant content and identify key information of IoT security flaws. After the collection of IoT vulnerability reports, we extract the semantics of vulnerability descriptions and other structured information (e.g., attack scripts) from the vulnerability reports. The descriptions here provide information about all circumstances under which a vulnerability can be exploited (e.g., all related parameters and locations), which enables us to leverage the attack surface observed from an attack script or other structured information like traffic logs to stop other attacks. A problem is that some of the vulnerability reports do not have vulnerability descriptions or structured information. Later we discuss how we handle this issue (see Section 4.3).

Architecture. Figure 2 illustrates the architecture of IoTShield, which has three major components: (1) data collection, (2) IoT vulnerability extraction, and (3) automated protection generation. Data collection is used to gather vulnerability reports from the Internet. The IoT vulnerability extraction is used to extract IoT vulnerability information from a large number of documents, including forums, blogs, and mailing lists. More specifically, we crawled popular online sources for bug disclosure and further ran a corpora quality analyzer to filter out the documents that are irrelevant to vulnerability reports. For the remaining documents, we used a recognizer to identify IoT vulnerability reports and extracted key information, such as their types, affected products, CVE numbers, authors, and published dates. This information will serve as the description for the later signature generation stage. Once given a set of IoT vulnerability reports, IoTShield first clusters the reports describing the same vulnerability together, and then utilizes NLP to discover the vulnerability semantics (e.g., vulnerability type, location, and parameters) from the descriptions. Then, it extracts the structured information (e.g., traffic logs, scripts, and Linux commands) to create an exploit (or PoC) template and find illegal parameters (e.g., those used to inject commands). After that, the signature generation component utilizes the vulnerability location and parameters to identify all related attack surfaces, which helps to determine all parameters for the template that can also lead to exploits. The parameters and template here form the signature. Finally, IoTShield automatically transforms the signature into the format used in existing IDSes or WAFs (the prototype built in our research outputs a Snort signature [37]) for a fast deployment.

Table 6: List of vulnerability reporting websites.

Categories	Website	Reports	IoT reports
Personal Blogs	s3cur1ty.de/advisories	28	16
	pierrekim.github.io	18	13
	gulftech.org	129	5
Forums	seclists.org/fulldisclosure	108,647	1,219
Team Blogs	coresecurity.com	390	31
	vulnerabilitylab.com	2,122	39
	blogs.securiteam.com	1,925	42
Mailing lists	seclists.org/bugtraq	85,593	1,591
Data Archive	exploit-db.com	39,380	895
	packetstormsecurity.com	97,093	1,951
	0day.today	30,177	834
	seebug.com	56,413	690
	myhack58	7,311	150
Total	-	42,9795	7,514

Dataset. As mentioned earlier, we ran our crawler across the vulnerability reporting websites listed in Table 6, including forums (`seclist.org/fulldisclosure`), mailing lists (`seclist.org/bugtraq`), personal blogs/advisories (`pierrekim.github.io`), research team advisories (`coresecurity.com`), and vulnerability archive websites (`packetstromsecurity.com`). These sources were collected from the external references included in CVEs, among which we selected the most frequently used ones. From these sources, we further manually picked out those related to IoT vulnerabilities, like `seclist.org`, and added them to the list, which also includes some research groups' websites known to report security vulnerabilities.

4.2 IoT Vulnerability Extraction

Preprocessing. Over the documents collected by the crawler, our IoT vulnerability extraction component removes the textual information irrelevant to vulnerabilities, such as advertisements, pictures, dynamical scripts, and navigation bar, while keeping the main content of each webpage with document URLs, document titles, authors, and publication dates. Since different websites have different templates and HTML structures, we manually analyzed each of them (13 vulnerability reporting sites in total, see Table 6) to identify useful content.

Corpora quality analyzer. After the preprocessing, we still need to filter out the documents irrelevant to IoT vulnerability reports, which is done as follows:

• *The percentage of dictionary words*. We removed the documents whose content contains mostly (above 82% in our research) dictionary words, which are recognized by enchant library [1], since a real vulnerability report always includes a significant amount of non-text information, like vulnerable paths and functions, PoC or scripts, etc. Otherwise, the text looks more like a survey article, white paper or notification.

• *The number of hyperlinks*. In general, a vulnerability report, particularly for IoT, is not supposed to include too many hyperlinks. Otherwise, it could be a summary for all the vulnerabilities disclosed, instead of a specific report with detailed vulnerability information. Thus, we discarded the documents with more than 25 hyperlinks.

•*Threshold justification*. The two threshold values above (i.e., 82% and 25) are based on our empirical experience, for the purpose of filtering out most non-IoT vulnerability reports with little collateral damage. To justify our threshold configuration, we attempt to estimate the possibility of a real IoT vulnerability report being wrongly discarded. To this end, we randomly sampled 100 documents being discarded and then manually examined whether there is any wrongly discarded case (i.e., a real IoT vulnerability report but discarded as non-IoT). What we found is that all of the sampled documents are irrelevant to IoT vulnerability reports (neither related to IoT vulnerability nor containing any vulnerability details). In the other words, no real IoT case exists among these 100 randomly sampled documents that are discarded as non-IoT. This implies that our empirical threshold configuration is very effective to filter out non-IoT vulnerability reports.

IoT vulnerability recognition. For the remaining documents, we further ran a recognizer to discover IoT vulnerability reports and extract key information (i.e., device type, vendor name, and vulnerability type). The retrieval of the vulnerability information is modeled as a named entity recognition problem [52] in NLP. More specifically, we first attempted to identify four IoT vulnerability-related entities, including device types, vendors, product names, and vulnerability types, and then utilize the dependency relationship across them to confirm the presence of vulnerability-related descriptions. Given the uniqueness of the descriptions, we adopted a set of IoT-specific recognition techniques to retrieve them, as elaborated on below.

Table 7: Context textual terms.

Entity	Context terms
Device Type	camera, ipcam, netcam, cam, dvr, router nvr, nvs, video server, video encoder, video recorder diskstation, rackstation, printer, copier, scanner switches, modem, switch, gateway, access point
Vendor	1,552 vendor names
Product	[A-Za-z]+[-]?[A-Za-z!]*[0-9]+[-]?[-]?[A-Za-z0-9] *∧[0-9]2,4[A-Z]+
Vuln type	733 CWE, 88 abbreviations
Version	(?:version[:.]*([\w-][\w.-]+) ve?r?s?i?o?n?s?[:.]*([\d-][\w.-]+)
CVE	CVE-[0-9]{4}-[0-9]{4,}

To identify these individual entities, we utilized keyword and regular expression based matching. For device types, vendor names, and vulnerability types, we used a set of keywords, as illustrated in Table 7: whenever a single word in the category (device type, vendor name, or vulnerability type) is found, we believe that its corresponding entity exists. These keywords are from the features of real-world devices. Specifically, we collected all common device types, including routers, cameras, modems, and printers, and found vendor names from Wikipedia. We also gathered vulnerability types from the Common Weakness Enumeration (CWE), which is a community-developed list of common software security weaknesses [9]. Further, we added to the list common acronyms of these vulnerabilities, such as CSRF and RCE. For product names, we built regular expressions to identify each entity, due to the large volume and difficulty of enumeration. These expressions are listed in Table 7.

In this way, our approach can identify all IoT vulnerability-related documents. However, given the pervasiveness of such entities (e.g., the term "switch" also appears in the documents unrelated to IoT flaws), using them alone could introduce a large number of false positives. To address this issue, we leveraged the dependency among these entities to ensure the correct recognition of these vulnerabilities.

Intuitively, when these entities are indeed used to describe an IoT vulnerability, they do not independently occur. Instead, they come together to present a concept. This implies the existence of dependency among them. In particular, the term for the vendor entity precedes the product entity or the device-type entity: e.g., D-Link DIR-600 or Foscam IPcamera. Also, the document needs to contain the vulnerability type, e.g., command injection. Using these rules, we can piece together these entities, linking IoT products to vulnerabilities. However, there are still several cases satisfying the rules above but as non-IoT device entities, such as "NAI NAI-0020" and "EE

Thomson TWG850 Wireless Router Multiple Vulnerabilities
Foscam IP Cameras Multiple Cross Site Request Forgery Vulnerabilities
Belkin Router N150 - Path Traversal Vulnerability
Dlink DIR-601 Command injection in ping functionality
Squirrelmail 1.4.22 Remote Code Execution
New Linux kernel 2.6.8 packages fix several vulnerabilities
Cisco Ironport Appliances - Privilege Escalation Vulnerability

Figure 3: Examples of the IoT vulnerability entities (first four) and non-IoT vulnerability entities (last three).

13EFF4". To suppress false positives, we further investigate the results by using an entity checker. More specifically, in the entity checker, we search for the extracted entities (e.g., D-Link DIR-600) in Google, and then calculate the cosine similarity between the extracted entities and the title of the search results. If the similarity is extremely low (e.g., 0.08), we regard the extracted entity as a non-IoT device.

Figure 3 shows examples of IoT vulnerabilities (the first four) and other vulnerabilities (the last three) in vulnerability reports. For each vulnerability extracting from the preprocessing stage, we used the keyword matching to identify the device type (e.g., router), vendor (e.g., Belkin), and vulnerability type (e.g., path traversal). Then, we used the regular expression based matching to extract product information (e.g., N150). After that, we checked if the extracted entities are combined to present a IoT vulnerability concept via local dependency and entity checker. For example, three entities "Belkin", "router", and "N150" together describe an IoT device. As we can see from Figure 3, the first four always include the affected IoT products (e.g., D-Link DIR-600) and the vulnerability types (e.g., command injection), while the latter three do not. Once such a report is found, our approach further extracts the firmware version and CVE number from its content when such information exists, for the follow-up analysis. The regular expressions for identifying these entities are also listed in Table 7.

4.3 Automatic Defense Rule Generation

After the IoT vulnerability recognition, we identified the entities (i.e., device types, vendors, product names, and vulnerability types) from the IoT vulnerability reports. These entities are then used to cluster IoT vulnerability reports describing the same IoT vulnerability. Then, given each cluster, we extracted the vulnerability semantics (e.g., vulnerability location[2] and exploit parameters) and other structured information (e.g., attack scripts) from the vulnerability reports. This generation process, as shown in Figure 4, enables us to leverage the attack surface to generate a vulnerability-specific signature.

[2]A vulnerability location is where flaws exist, such as "command injection in PwdGrp.cgi", "PwdGrp.cgi" is the vulnerability location.

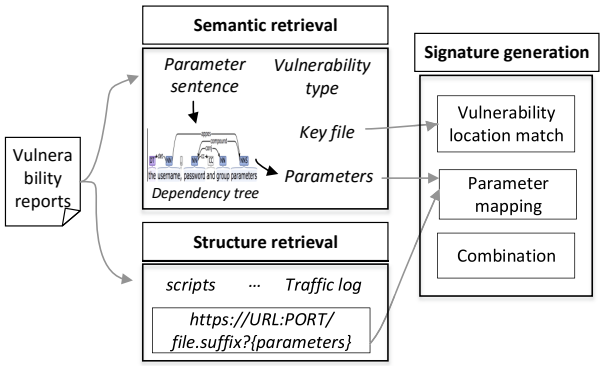

Figure 4: The architecture of signature generation.

Figure 5 presents an example, showing how we generate a signature from an extracted IoT vulnerability report. On the top-left of the figure is the vulnerability description. The bottom-left is the content of the structured information, which is a traffic log for an exploit on this vulnerability. IoTShield first locates the vulnerability semantics in the description. For instance, the sentence "command injection in PwdGrp.cgi" indicates that the vulnerability type is "command injection" and the affected location is "PwdGrp.cgi", together with the vulnerable parameters from another sentence "the username, password, and group parameters". Then, IoTShield parses the structured information (e.g., the traffic logs), and discovers the path of the vulnerable CGI "/cgi-bin/supervisor/PwdGrp.cgi" and the parameter used for command injection "pwd = ;reboot;", from the command indicator ";" and the list of legitimate commands.

Further, from the sentence about other vulnerable parameters "the username, password, and group parameters" (from the description), we can now infer that the same injection can also happen on "usr" and "grp", but not on the parameters "action" and "lifetime". In this way, we can build a vulnerability-specific signature for the injection flaw, and then transform it into the Snort format (based on vulnerability type) as presented in Figure 5. Here we elaborate on the individual components of IoTShield.

Report clustering. There are some different blogs describing the same vulnerability, and these reports may describe the vulnerability from different aspects. So, we need to cluster these reports together to form a complete vulnerability report before the rule generation stage. The challenge is that although two reports describe the same vulnerability, they may have different formats with different hash values. Our solution is to use the entities (i.e., device types, vendors, product names, and vulnerability types), which are recognized from IoT vulnerability reports, to cluster IoT vulnerability reports that describe the same IoT vulnerability.

Note that report clustering aims to supplement the missing information of a vulnerability report (e.g., missing script code or missing vulnerability description). There rarely exist

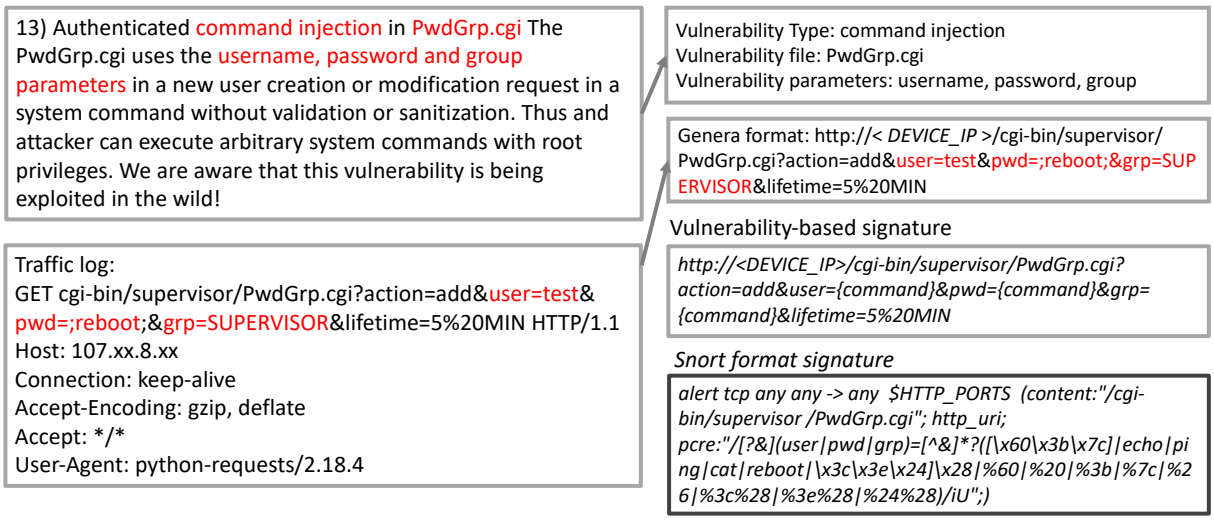

Figure 5: Example of vulnerability-specific signature generation from a vulnerability report.

different vulnerabilities with same device type, vendor, product name, and vulnerability type. However, even different vulnerabilities are clustered together, we can extract signatures for each vulnerability when matching the template (see Section 4.3)

Semantic and structured information retrieval. We first utilized NLP techniques to analyze the vulnerability descriptions to find vulnerability semantics, including vulnerability type, location, and short sentences with exploit parameters. Our approach is based on the observation that the semantic information of interest is presented through a relatively stable grammatical structure. Specifically, we utilized the vulnerability type list in Section 4.2 to determine the type of the problem documented and the regular expressions to find the vulnerability location, which should be a web content file, such as ".htm", ".cgi", ".php", ".asp", and ".html". When it comes to vulnerability parameters, our approach locates the sentences containing the keyword "parameter", "variable", "action", or "function", and leverages the grammatical relationship among these words and the values of the parameters to find them. For this purpose, we constructed a dependency tree using the Stanford Dependency Parser [51] to parse the whole sentence and then extract the nouns as the targeted terms to inspect their relationship with the keywords. These terms are considered to be parameters if they have a non-"nmod" relation [3] with the keyword, or have a "conj" relation [4] with an identified parameter. An example is shown in Figure 6.

Further, we used the regular expressions (listed in Table 13 of Appendix) to locate different kinds of the structured information, including the PoC using Linux commands (curl and wget), PoC URL, PoC HTML scripts, and PoC traffic log, etc. For each type of the structured information, we built a parser

[3]One element serves as a nominal modifier for the other.
[4]Two elements are connected by a coordinating conjunction.

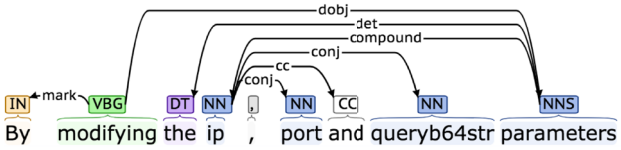

Figure 6: The dependency tree of a sentence with vulnerability parameters.

to transform it into a general template:

$$http://HOST:PORT/file.suffix?\{parameters\},$$

where HOST is the device's IP address, port is the application layer server port (default is 80), file.suffix is the vulnerability location, and {parameters} is the key-value format that includes the parameters used for the vulnerability file in the exploits. For each item in {parameters}, IoTShield checks whether it carries any illegal values like injected Linux command or Java scripts. Note that not all vulnerabilities need parameters, such as some information disclosure weaknesses, which could be exploited by simply requesting the vulnerable location. In this way, our approach acquires the semantic and structured information from the vulnerability reports.

Signature generation. After discovering vulnerability information from the reports, IoTShield utilizes it to build a signature. Specifically, we first compared the vulnerability location (recovered from the description) and "`file.suffix`" in the template (from the structured information); if matched, we believed that the semantic information (including vulnerability types and parameters) would be about the flaw modeled by the template (e.g., that attacked by the script). Then, based on the vulnerability type, we decided whether to ignore the parameter part of the template, since some vulnerabilities do not need parameters to exploit (e.g., information disclo-

sure and directory traversal). For the vulnerability types that do not need parameters to exploit (e.g., directory traversal ("/../../etc/passwd") in D-Link routers [CVE-2018-10822]), we ignore the parameter part. As of all the vulnerability types in our data, information disclosure and directory traversal are the only two typical vulnerability types not requiring a parameter to exploit. For those vulnerabilities that need parameters, IoTShield generalizes the parameter field in the template with those collected from the description. In this way, we built a vulnerability-specific signature. An example is shown in Figure 5.

A problem with this simple signature generation process is that the semantic information and the structured information may not always match in an exact fashion. For example, in Figure 5, the vulnerability parameters in the vulnerability description are presented in the natural language (i.e., username, password, and group) while they are abbreviated in the structured information (i.e., user, pwd, and grp). To address this problem, we manually collected a list mapping individual keywords to their corresponding abbreviations used in Linux, such as grp for group, for translating natural language terms into parameters in a signature.

Another issue is that, as mentioned earlier, some vulnerability reports do not contain both vulnerability descriptions and structured information; or vulnerability descriptions do not contain all vulnerability semantics. Next, we discuss how to handle them.

• *Vulnerability description only*. Without structured information, all we can get are just the vulnerability type, locations, and parameters. The parameters, however, can be less reliable due to the abbreviations they could have, which we cannot see. Therefore, we can only generate signatures for some vulnerabilities: those in which the knowledge of the vulnerability location alone (e.g., "PwdGrp.cgi") is enough for protection. Examples of such flaws include information disclosure and directory traversal, e.g., for an information disclosure problem, a request for `/QIS_wizard.htm` is sufficient for signature generation.

• *Structured information only*. Without vulnerability description, we cannot produce a generalized, vulnerability-specific signature. However, we can still utilize the structured information to build an exploit-specific signature to defeat the attack script described in a vulnerability report. As found in our measurement study (Section 3), today's IoT attacks often utilize these published scripts. These signatures can still contribute to the detection of many ongoing attacks, though they can be evaded once an adversary is willing to make more effort to better understanding the flaws he attacks. It is important to note that some level of generalization is still possible here; for example, once a parameter recovered from an attack script is found to contain a Linux command, we can add other commands commonly used in attacks to the parameter list for signature generation.

5 Implementation and Deployment

5.1 Implementation

We implemented a prototype system of IoTShield, which includes a set of building blocks. Here we briefly describe the system's nuts and bolts, and then show how they are assembled into the prototype system.

Nuts and bolts. Our prototype system was built upon three key functional components: *report crawler*, *vulnerability extractor*, and *rule generator*. Those components are extensively used across the whole system, and they were implemented as follows.

• The report crawler fetches the vulnerability reports from the Internet using wget and the scrapy crawling framework [36]. Specifically, for some websites that are well archived (e.g., `seclists.org/fulldisclosure`), we used "wget –mirror" to download their pages recursively (i.e., crawl the websites as deep as possible). For other websites, we used the scrapy crawling framework to crawl the whole websites. Since websites sometimes have crawling restrictions (e.g., `packetstorm.com` has a rate limit and `s3cur1ty.de` requires a cookie in the header of each request), our crawler simulates browser behaviors to mitigate those restrictions. It utilizes different user-agents for each request and sleeps for a random period of time after sending out multiple requests; additionally, once an access fails, a new attempt will be made later, after all pending requests in the current waiting queue have been delivered.

• The vulnerability extractor was implemented by 2,300 lines of python code. The Beautiful Soup Python library [5] was used to parse the vulnerability reports to extract main contents. The NLTK [29] package was used to split sentences, stem words, remove stop words, etc. The Aho–Corasick algorithm [2] is a string-searching algorithm to speed up the entity identification stage. A scikit-learn [35] library was used to calculate the TF-IDF (Term Frequency-Inverse Document Frequency) cosine similarity in the entity checker.

• The rule generator was implemented by 1,500 lines of Python code. We used a Simhash Algorithm [31] to detect near-duplicates, and the Stanford dependency parser [51] to establish the dependency tree.

5.2 Deployment

IoTShield can be deployed in two modes: coarse-grained and fine-grained. In the coarse-grained mode, all the generated rules are used in the IDS system, regardless of device types. All the rules are used to inspect the network traffic. This mode is easy to deploy but may have some false positives, since some rules can be device-specific. Also, in the coarse-grained mode, we suggest not to use the rules generated from descriptions only. This is because when ignoring device type, the rules generated from descriptions only may lead some false

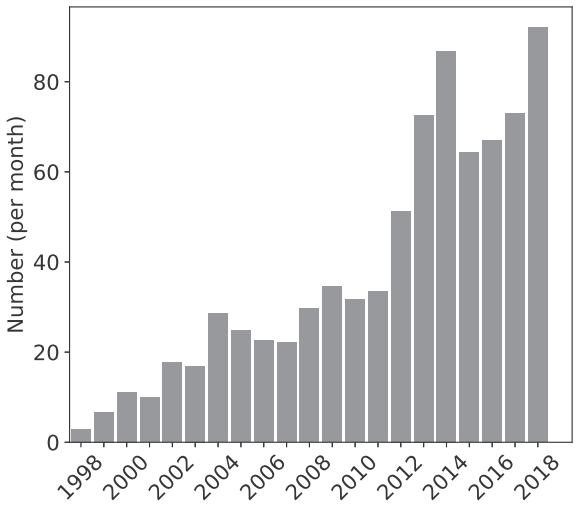

Figure 7: Vulnerability disclosure trend.

positives. For example, for a report describing an information disclosure vulnerability at the file of "/new/index.htm" in Merit Lilin IP Cameras, with description only, we generate a signature to block the traffic that attempts to extract information from the file of "/new/index.htm". However, such a scenario (retrieving the file of "/new/index.htm") may also occur in normal web servers and cause false positives. In the fine-grained mode, we take the network environments into account and deploy rules for given device types. For example, if there are just D-link devices in the local network, we only deploy the rules to protect D-link vulnerabilities. More specifically, in the fine-grained mode, IoTShield first analyzes the network traffic or actively probes the network to identify device types (e.g., models and brands) and their IP addresses. This step can be achieved by using a device list in the monitored network or using the device fingerprinting method proposed in [47]. After that, a signature selection process will be conducted to select the corresponding signatures, given the current device types in the network.

6 Evaluation

6.1 Effectiveness

To validate the efficacy of IoTShield, we first manually checked the extracted IoT vulnerabilities and obtained some basic statistics of them. Then, we used two different traffic traces to evaluate the effectiveness of generated signatures.

Vulnerability extractor. We randomly sampled 200 reports from those identified for manual validation and achieved a precision of 94%. In total, we collected 7,514 IoT vulnerability reports from 0.43 million articles (Table 6). These reports disclose 12,286 IoT vulnerabilities, with roughly 1.6 each on average. Figure 7 shows the average number of IoT vulnerabilities disclosed per month from 1998 to 2018. We can see that the number of disclosures has increased since 1998, and this increasing trend has further sped up since 2012 and slowed down since 2014, but it peaked in 2018 when about 90 IoT vulnerabilities per month were disclosed.

Table 8: List of top 10 vendors and device types of affected devices.

Device Vendor	Num	Device Type	Num
Cisco	1,264	router	3,700
D-Link	988	switch	1,422
Linksys	539	camera	1,248
Netgear	522	firewall	1,101
HP	485	gateway	1,032
Symantec	299	modem	843
TP-Link	255	access point	478
Zyxel	229	printer	408
Huawei	195	nas	338
Asus	180	scanner	176

Table 9: List of top 10 vulnerability types.

	Vulnerability type	Num
1	Denial of service	975
2	CSRF	902
3	Buffer overflow	869
4	Command injection	806
5	XSS	775
6	Authentication bypass	763
7	Command execution	458
8	Information disclosure	407
9	Directory traversal	307
10	Privilege escalation	276

These IoT vulnerabilities are related to device types and vendors. We found that the distribution of vulnerabilities among IoT vendors follows a long-tail: nearly 60% of vulnerable devices are from the top 10 vendors with the most security flaws. Table 8 lists the vulnerability distribution over these 10 device types and vendors. As we can see here, routers, switches, and cameras, which are perceived to be the most common IoT devices, also have the most vulnerabilities. In addition, the vendors responsible for the most vulnerabilities (i.e., Cisco, D-Link, and linksys) are all reputable and have the largest market shares.

Table 9 further lists the top 10 vulnerability types in our dataset. The majority of them are remotely exploitable (e.g., buffer overflow, denial-of-service, CSRF command injection, and authentication bypass), which could be easily used to compromise IoT devices. Moreover, cross-site scripting (XSS), command injection and command execution are commonly used by IoT malware to execute commands on a compromised device as a botnet node.

Table 10: Effectiveness of IoTShield.

Dataset	Precision	Recall	False Positive Rate
Real devices	97%	83%	0.01%
Honeypot	98%	93%	0.06%

Table 11: Running time at different stages.

Stage	Running time (s)	Percentage
Data collection	0.386	51%
IoT vulnerability extraction	0.154	21%
Rule generation	0.210	28%
Overall	0.750	100%

Rule generation effectiveness. We first evaluated our IoTShield prototype on 190K HTTP requests collected from IoT devices and honeypots, using a Macbook Pro with 2.6GHz Intel Core i7 and 16GB of memory. Again, all signatures generated were in the Snort format.

Our HTTP requests include those gathered from real-device honeypots and those from the simulators. In our experiments, we labeled IoT device traffic as described in Section 3.1. Those traces include 178,778 HTTP requests received by the simulators, which are related to 141 attack activities generated by 26 unique attack scripts, and the rest is benign traffic. The remaining data come from the real-device honeypots, as described in Section 3.1, including 11,602 HTTP requests in 1,860 attacks generated by 81 unique attack scripts.

We evaluated the effectiveness of IoTShield using precision, recall, and the false positive rate (FPR). Precision is defined as $|TP|/|FP+TP|$, recall is $|TP|/|TP+FN|$, and FPR is $|FP|/|FP+TN|$, where TP is the number of true positives, FN is the number of false negatives, FP is the number of false positives, and TN is the number of true negatives. Table 10 presents the experimental results. Over the traces received by the simulators, the precision of our automatically generated signatures is 98%, the recall is 93%, and the FPR is 0.06%. Over the requests gathered from the real devices, our signatures can achieve a 97% precision, 83% recall, and 0.01% FPR.

We further used a long-time traffic captured in an industrial control system's HMI honeypot for the evaluation of IoTSheild. The simulated industrial control system's HMI honeypot is used to monitor the attack traffic with a blind scanning and attack. The duration was from October 2017 to November 2018 across from seven different cities. By replaying the traffic, IoTShield reported 7,396 alerts of exploiting the HMI system. By manually checking the 7,396 alerts, we confirmed that about 6,705 alerts were indeed IoT attacks. The rest of the alerts were confirmed to have attacked other vulnerabilities on common web servers. For instance, "`/level/77/exec/show/config/cr`", is found in exploit-db as a script to evade detection of HTTP attacks via non-standard "%u" Unicode encoding of ASCII characters in the requested URL. [7].

6.2 Performance

Signature generation. To understand the performance of IoTShield, we conducted experiments to measure the time cost of processing vulnerability reports at each individual stage: data collection, IoT vulnerability extraction, and automated rule generation. The IoTShield prototype runs on a commercial desktop computer (Ubuntu 18.04, 8GB of memory, 64-bit OS, with 4-core Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz), indicating that the CPU and memory requirements of IoTShield can be easily met. The IoTShield process runs in a single thread. Table 11 lists the average time cost of each stage of IoTShield for one rule generation. The acquisition of vulnerability reports from the Internet takes 0.386 seconds (51%). Note that this stage requires message transmissions, and the time cost is dependent upon the network conditions. The IoT vulnerability extraction takes 0.154 seconds (21%), while the rule generation costs more time, 0.21 seconds (28%), due to the fact that it needs to establish the dependency tree of the sentence with vulnerability parameters. Overall, the time cost of IoTShield for automatic rule generation is low in practice, and we could further reduce the time cost by running it in multiple threads. The results indicate that IoTShield is efficient and can be easily scaled to a desirable level to handle the massive amounts of vulnerabilities online with a timely update of the defense rules.

Rule inspection. To further evaluate the performance of IoTShield in practice, we ran it as one component of an IDS for processing the real-world traffic captured on the edge router of a research institution, which consists of more than 100,000 Internet devices. The amount of traffic is about 53G, and the duration of this traffic collection is about two hours. We replayed the traffic to Snort with and without IoTShield. For the Snort without IoTShield, it costs 426.28 seconds to inspect all the collected packets; for the Snort with IoTShield, it only adds 0.13 seconds for rule inspection over the entire 53G of data, showing that IoTShield induces little overhead to IDSes for online data processing.

7 Discussion

Limitation. In the data collection, we crawled 13 different websites retrieving 0.4 million reports, and we plan to keep a periodic update in the future. However, we acknowledge that our methods cannot exhaustively collect all IoT vulnerabilities in the wild. Although we believe that we have collected the majority of IoT vulnerabilities in public, other methods

for data collection could also be considered to collect those vulnerabilities recorded in less popular websites, such as personal blogs or social networking logs. For example, since IoT vulnerabilities are usually targeted for some specific devices, we will search product names combined with vulnerability types as query keywords in search engines to crawl more reports.

In addition, although we did observe that the majority (80%) of the malicious HTTP requests are from blind scannings, not targeted at specific IoT devices, we acknowledge that the traffic logs collected by the honeypots could be biased, due to the IoT device types in the honeypots and deployment time periods of the honeypots. Ideally, IoTShield could be deployed in ISPs for a large-scale and real-scenario evaluation. However, we were unable to access ISP data. Alternatively, we performed a real-scenario evaluation based on the traffic we captured from edge routers of a research institution, whose data size is about 114G and the duration is about 12 hours. In this experiment, IoTShield only produced two false positive alerts; considering the substantial data size, the FPR is close to zero.

Since IoTShield deals with generally uncoordinated sources of vulnerability information, it may sometimes face incomplete source information (missing both vulnerability descriptions and structured information, or vulnerability descriptions not containing all vulnerability semantics), even we have used report clustering to supplement the missing information. As mentioned in Section 4.3, we indeed take the problem of incomplete information into serious considerations. For structured information only (about 9% observed in our dataset), we generate an exploit-specific signature that can contribute to the detection of many ongoing attacks. However, we acknowledge that we may miss some exploit variants, which leads to the decrease of recall. For vulnerability description only (about 20% observed in our dataset), we use vulnerability location alone (e.g., "PwdGrp.cgi") as signature for some specific vulnerabilities, which may lead to the increase of false positive rate. In this way, IoTShield can only generate signatures for limited vulnerability types (e.g., information disclosure and directory traversal). A natural follow-up step is to investigate the missing information and explore a systematic information supplement method. We will leave this as our future work.

Also, IoTShield cannot handle the exploits in some specific program languages, and its processing capability is limited to traffic logs, scripts, Linux commands, etc. This is because IoTShield cannot easily generate the general exploit template to identify vulnerability locations and parameters in a different program language. In our future work, we plan to develop a simulation system to execute these programs and generate the attack traffic. Thus, we will be able to produce the attack script in the traffic format (e.g., HTTP request) for whichever program language is used. Moreover, although we did observe a few IoT vulnerabilities in other application layer protocols, our defense only targets HTTP vulnerabilities, which cover most (90%) of the IoT vulnerabilities we observed in the honeypots and vulnerability reports. In our future work, we plan to cover more vulnerabilities, which exploit other application layer protocols, by extracting vulnerability semantic and structure information based on individual application layer protocol's domain knowledge.

Mitigation. Based on the results of our measurement study, we have identified several potentially effective mitigation strategies to restrain the fast-growing IoT-based attacks. In our study, we observed a large number of vulnerability reports in the wild, which are missed by vendors but exploited by attackers. We also observed the heterogeneity of IoT devices. IoT devices usually do not have an automatic update mechanism and are maintained by device users who may lack security awareness. Thus, the mitigation of an IoT-based attack requires a collaboration among users, vendors, and security researchers.

First, vendors should provide an official vulnerability report platform, and reply to vulnerability reports in a timely manner. In this work, we observed a relatively short duration of public disclosure: report authors usually disclosed the bugs after contacting vendors three times if no reply is received. In addition, vendors are expected to provide technical support for their discontinued products, especially those devices that are still widely used. Since most users lack security awareness, a vendor should increase efforts to avoid misconfiguration and notify these users about updating their devices in a more effective fashion (e.g., using an automatic update mechanism). Second, the authors of vulnerability reports should follow the guidelines for vulnerability reporting, such as a coordinated disclosure. We observed that at least 2,000 vulnerabilities were released before the vendors provided patches. Even worse, some report authors released a disclosure without attempting to contact the vendors or CVE. Finally, device users should pay more attention to the device configuration (e.g., default password) and quickly update vulnerable firmware when a new version becomes available.

Ethical issues. One ethical concern is the way by which the security reports were gathered, i.e., scraping various websites. We deployed certain mechanisms to bypass rate limiting and authentication. However, our process follows the robot exclusion protocol (robots.txt) of websites and causes no harm to them or their users. Another ethical concern is that we purchased attack tools on the black market. In our research, we consulted with our Institutional Review Board (IRB) (though no IRB review is required) and legal consul to ensure that the purchase has been done within the legal and ethical boundaries. The purchases did not violate any law and regulation. Also during the process, we refrained from gathering any information not supposed to collect, such as identity-related data.

8 Related Work

Vulnerability-specific signature generation. Cui et al. [46] presented a system for automatically generating a vulnerability-specific signature (or data patch) for an unknown vulnerability, given a zero-day attack instance. Their system injects the softwares/real devices to generate the variants of attack instances. However, it cannot be easily applied for the IoT vulnerability-specific signature generation, due to the large amount of different IoT device vulnerabilities being covered. Wang et al. [60] proposed a vulnerability-specific network filter, Shield, at an end-system to prevent known vulnerability exploits. Shield requires a manually-generated policy to describe the vulnerability. Specially, it requires a fairly deep understanding of the protocol over which the vulnerability is exploited. It is not acceptable for the IoT vulnerability signature generation, due to the significant amount of manual efforts to generate policies for each IoT device. Brumley et al. [44] proposed data-flow analysis techniques for automatically generating vulnerability-specific signatures. However, it cannot be deployed in IoT vulnerability signature generation, due to the lack of source code. By contrast, IoTShield analyzes the content of more than 7,500 IoT vulnerability reports and recovers key knowledge to generate vulnerability-specific signatures.

NLP for vulnerability assessment. Pandita et al. [54] used NLP techniques to analyze Android APP descriptions and API documents for determining unnecessary permissions. Sabottke et al. [55] explored the vulnerability-related information disseminated on Twitter and provided an early warning for the existence of real-world exploits by tweets. Liao et al. [50] presented a novel technique for automatic Indicator-of-Compromise extraction from unstructured text. You et al. [61] proposed leveraging vulnerability-related text (CVE reports and Linux git logs) to guide Linux kernel vulnerability fuzzing. Zhu et al. [62] mined Android documents and security literature to generate features for detecting Android malware. Caselli et al. [45] proposed to automatically mine parameter configuration rules of network control systems (e.g., BACnet-based building automation systems) from system specifications. In contrast to these previous works, we utilize a set of IoT vulnerability reports' syntax features to discover vulnerability-specific knowledge for IDS signature generation to protect IoT from being attacked.

Vulnerability-related measurement. Shahzad et al. [58] conducted a large-scale study on the software vulnerability life-cycle based on public vulnerability databases. They utilized association rule mining to extract the relationship between the representative exploitation behavior of hackers and the patching behavior of vendors. Nappa et al. [53] presented a systematic study of patch deployment in client-side vulnerabilities, in order to analyze how users deploy patches. They found that the patch mechanism has an important impact upon the patch deployment rate. Li et al. [48] performed an extensive study on the effectiveness of vulnerability notifications, with the aim of illuminating which fundamental aspects of notifications have the greatest impact. Li et al. [49] conducted a large-scale empirical study of security patches based on the open-source software projects. They sought to identify the differences between security and non-security bug fixes. Sarabi et al. [56] studied the vulnerability patching by analyzing vulnerabilities across four software products. Their focus is mainly on how individual behaviors influence the security state of an end-host. In contrast to previous works focusing on vulnerabilities in CVE or NVD, our work extensively studies the IoT-related vulnerabilities that are from a large number of vulnerability reports scattered around forums, mailing lists, and blogs, and we further explore the effectiveness of using such reports for IoT vulnerability defense.

9 Conclusion

To understand how cybercriminals launch IoT-related attacks, we leveraged honeypots to collect the traces of real-world IoT exploits and analyzed four popular attack toolkits. Our research sheds light on a largely overlooked cause of the pervasiveness of IoT attacks in recent years: IoT vulnerabilities are publicly available and easy to exploit, and today's IoT attacks almost exclusively use known vulnerabilities for mounting malicious attacks. More importantly, our findings lead to the design of IoTShield, a simple yet effective IoT vulnerability-specific signature generation system for intrusion detection. IoTShield first collects 430,000 vulnerability reports from the past 20 years and identifies content of 7,500 IoT vulnerability reports. IoTShield then retrieves key knowledge to generate vulnerability-specific signatures. These signatures can be easily deployed at existing intrusion detection systems or web application firewalls to detect exploit attempts on a target IoT device. Therefore, IoTShield significantly raises the bar for future IoT attacks to succeed.

Acknowledgments

We are grateful to our shepherd Adwait Nadkarni and anonymous reviewers for their insightful feedback. We also want to thank Haoran Lu and Jianzhou You for help collecting underground attack tools and honeypot data. The IIE authors are supported in part by National Key R&D Program of China (No. 2018YFB0803402), Key Program of National Natural Science Foundation of China (No. U1766215), and International Cooperation Program of Institute of Information Engineering, CAS (No. Y7Z0451104). The IU authors are supported in part by NSF CNS-1850725, 1527141, 1618493, 1838083 and 1801432 and ARO W911NF-16-1-0127. The support provided by China Scholarship Council (CSC) during a visit of Xuan Feng to IU is acknowledged.

References

[1] Abiword - Enchant. http://www.abisource.com/projects/enchant/.

[2] Aho-corasick algorithm. https://github.com/WojciechMula/pyahocorasick/.

[3] Amnesia. https://researchcenter.paloaltonetworks.com/2017/04/unit42-new-iotlinux-malware-targets-dvrs-forms-botnet/.

[4] AVTECH IP Camera, NVR, DVR multiple vulnerabilities. http://seclists.org/fulldisclosure/2016/Oct/36.

[5] Beautifulsoup. https://www.crummy.com/software/BeautifulSoup/.

[6] BrickerBot. https://www.trustwave.com/Resources/SpiderLabs-Blog/BrickerBot-mod_plaintext-Analysis/.

[7] Cisco Secure IDS 2.0/3.0 / Snort 1.x / ISS RealSecure 5/6 / NFR 5.0 - Encoded IIS Detection Evasion. https://www.exploit-db.com/exploits/21100.

[8] CNVD-2013-20783. http://www.cnvd.org.cn/webinfo/show/3205.

[9] Common Weakness Enumeration. https://cwe.mitre.org/.

[10] CVE-2009-1557. https://nvd.nist.gov/vuln/detail/CVE-2009-1557.

[11] CVE-2009-1558. https://nvd.nist.gov/vuln/detail/CVE-2009-1558.

[12] CVE-2009-1559. https://nvd.nist.gov/vuln/detail/CVE-2009-1559.

[13] CVE-2009-1560. https://nvd.nist.gov/vuln/detail/CVE-2009-1560.

[14] CVE-2013-2578. https://nvd.nist.gov/vuln/detail/CVE-2013-2578.

[15] CVE-2013-2579. https://nvd.nist.gov/vuln/detail/CVE-2013-2579.

[16] CVE-2013-2580. https://nvd.nist.gov/vuln/detail/CVE-2013-2580.

[17] CVE-2013-2581. https://nvd.nist.gov/vuln/detail/CVE-2013-2581.

[18] CVE-2013-3612. https://nvd.nist.gov/vuln/detail/CVE-2013-3612.

[19] CVE-2013-3688. https://nvd.nist.gov/vuln/detail/CVE-2013-3688.

[20] CVE-2017-16957. https://nvd.nist.gov/vuln/detail/CVE-2017-16957.

[21] D-Link DIR-645 Routers Remote Authentication Bypass Vulnerability. https://www.securityfocus.com/bid/58231.

[22] Guidelines for Security Vulnerability Reporting and Response . https://www.symantec.com/security/OIS_Guidelines%20for%20responsible%20disclosure.pdf.

[23] Hacking the D-Link DIR-890L. http://www.devttys0.com/2015/04/hacking-the-d-link-dir-890l/.

[24] Hajime. http://blog.netlab.360.com/hajime-status-report-en/.

[25] IoT reaper. https://krebsonsecurity.com/2017/10/reaper-calm-before-the-iot-security-storm/.

[26] Masuta. https://blog.newskysecurity.com/masuta-satori-creators-second-botnet-weaponizes-a-new-router-exploit-2ddc51cc52a7.

[27] Multiple vulnerabilities found in Wireless IP Camera (P2P) WIFICAM cameras and vulnerabilities in custom http serve. https://pierrekim.github.io/blog/2017-03-08-camera-goahead-0day.html.

[28] Multiple Vulnerabilities in TP-Link TL-SC3171 IP Cameras. https://www.coresecurity.com/advisories/multiple-vulnerabilities-tp-link-tl-sc3171-ip-cameras.

[29] Natural language toolkit. http://www.nltk.org/.

[30] Pwning CCTV cameras . https://www.pentestpartners.com/security-blog/pwning-cctv-cameras/.

[31] A python implementation of simhash algorithm. https://leons.im/posts/a-python-implementation-of-simhash-algorithm/.

[32] Remote Code Execution in CCTV-DVR affecting over 70 different vendors. http://www.kerneronsec.com/2016/02/remote-code-execution-in-cctv-dvrs-of.html.

[33] Reverse SSH tunneling. https://askubuntu.com/questions/598626/direct-ssh-tunnel-through-a-reverse-ssh-tunnel.

[34] Satori. https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/source-code-of-iot-botnet-satori-publicly-released-on-pastebin.

[35] Scikit-learn machine learning in python. http://scikit-learn.org/stable/index.html.

[36] Scrapy: A fast and powerful scraping and web crawling framework. https://scrapy.org.

[37] Snort - Network Intrusion Detection & Prevention System. https://www.snort.org/.

[38] SSH Port forwarding. https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding.

[39] The Internet of Things Will Be Even More Vulnerable to Cyber Attacks. https://www.chathamhouse.org/expert/comment/internet-things-will-be-even-more-vulnerable-cyber-attacks.

[40] Ross Anderson. Security in open versus closed systems—the dance of boltzmann, coase and moore. Technical report, Cambridge University, England, 2002.

[41] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *Proceedings of the USENIX Security Symposium*, pages 1093–1110, 2017.

[42] William A Arbaugh, William L Fithen, and John McHugh. Windows of vulnerability: A case study analysis. *Computer*, 33(12):52–59, 2000.

[43] Thanassis Avgerinos, Sang Kil Cha, Alexandre Rebert, Edward J Schwartz, Maverick Woo, and David Brumley. Automatic exploit generation. *Communications of the ACM*, 57(2):74–84, 2014.

[44] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. Towards automatic generation of vulnerability-based signatures. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 2–16, 2006.

[45] Marco Caselli, Emmanuele Zambon, Johanna Amann, Robin Sommer, and Frank Kargl. Specification mining for intrusion detection in networked control systems. In *Proceedings of the USENIX Security Symposium*, pages 791–806, 2016.

[46] Weidong Cui, Marcus Peinado, Helen J Wang, and Michael E Locasto. Shieldgen: Automatic data patch generation for unknown vulnerabilities with informed probing. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 252–266, 2007.

[47] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. Acquisitional rule-based engine for discovering internet-of-thing devices. In *Proceedings of the USENIX Security Symposium*, pages 327–341, 2018.

[48] Frank Li, Zakir Durumeric, Jakub Czyz, Mohammad Karami, Michael Bailey, Damon McCoy, Stefan Savage, and Vern Paxson. You've got vulnerability: Exploring effective vulnerability notifications. In *Proceedings of the USENIX Security Symposium*, pages 1033–1050, 2016.

[49] Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 2201–2215, 2017.

[50] Xiaojing Liao, Kan Yuan, XiaoFeng Wang, Zhou Li, Luyi Xing, and Raheem Beyah. Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 755–766, 2016.

[51] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[52] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[53] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 692–708, 2015.

[54] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. Whyper: Towards automating risk assessment of mobile applications. In *Proceedings of the USENIX Security Symposium*, pages 527–542, 2013.

[55] Carl Sabottke, Octavian Suciu, and Tudor Dumitras. Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In *Proceedings of the USENIX Security Symposium*, pages 1041–1056, 2015.

[56] Armin Sarabi, Ziyun Zhu, Chaowei Xiao, Mingyan Liu, and Tudor Dumitraş. Patch me if you can: A study on the effects of individual user behavior on the end-host

vulnerability state. In *Proceedings of the International Conference on Passive and Active Network Measurement*, pages 113–125. Springer, 2017.

[57] E Eugene Schultz Jr, David S Brown, and Thomas A Longstaff. Responding to computer security incidents: Guidelines for incident handling. Technical report, Lawrence Livermore National Lab., CA (USA), 1990.

[58] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X Liu. A large scale exploratory analysis of software vulnerability life cycles. In *Proceedings of the IEEE International Conference on Software Engineering*, pages 771–781, 2012.

[59] Shodan. The search engine for Internet-connected devices. https://www.shodan.io/.

[60] Helen J Wang, Chuanxiong Guo, Daniel R Simon, and Alf Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In *Proceedings of the ACM SIGCOMM Computer Communication Review*, pages 193–204, 2004.

[61] Wei You, Peiyuan Zong, Kai Chen, XiaoFeng Wang, Xiaojing Liao, Pan Bian, and Bin Liang. Semfuzz: Semantics-based automatic generation of proof-of-concept exploits. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 2139–2154, 2017.

[62] Ziyun Zhu and Tudor Dumitras. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 767–778, 2016.

A Keywords and Regular Expressions

Table 12: Keywords used in finding attack tools.

Keywords
Amnesia, next-gen Mirai botnets , Mirai Okiru, IoTroop
Satori IoT botnets, Brickerbot IoT botnets, Masuta,
Tsunami, Reaper, Hajime, IoT Mine Bitcoin, IoT hacking tool,
Persirai Botnet, Bashlite, Gafgyt, Qbot, Remaiten, Torlus,
DoubleDoor IoT botnet, JenX, IP Cameras GoAhead

Table 13: Regular expressions of structured information.

	Regular expressions
Linux command	curl [\w\W]+?\n\n
	wget [\w\W]+?\n\n
	wget [\w\W]+?\n
	curl [\w\W]+?\n)
Traffic log	POST [\w\W]+?\n\n
	GET[\n][\w\W]+?\n\n
	PUT [\w\W]+?\n\n
	POST [\w\W]+?\n\n [\w\W]+?\n
Exploit URL	((https?)://?[-A-Za-z0-9+&@#/%?\
	*=≃&$\\‘’\" \[\]() <>\]!:,.;]+[-A-Za-z0-9 +
	\[\]&’\" ()\]*<># \\/%= _])
Timeline	timeline[\w\W]+? \n\n[\-_]?
	time line[\w\W]+? \n\n[\-_]?