No Spring Chicken: Quantifying the Lifespan of Exploits in IoT Malware Using Static and Dynamic Analysis

Arwa Abdulkarim Al Alsadi
Delft University of Technology
Delft, The Netherlands
a.a.a.alalsadi@tudelft.nl

Kaichi Sameshima
Yokohama National University
Yokohama, Japan
sameshima-kaichi-mx@ynu.jp

Jakob Bleier
TU Wien
Vienna, Austria
jakob.bleier@tuwien.ac.at

Katsunari Yoshioka
Yokohama National University
Yokohama, Japan
yoshioka@ynu.ac.jp

Martina Lindorfer
TU Wien
Vienna, Austria
mlindorfer@iseclab.org

Michel van Eeten
Delft University of Technology
Delft, The Netherlands
M.J.G.vanEeten@tudelft.nl

Carlos H. Gañán
Delft University of Technology
Delft, The Netherlands
C.HernandezGanan@tudelft.nl

ABSTRACT

The Internet of things (IoT) is composed by a wide variety of software and hardware components that inherently contain vulnerabilities. Previous research has shown that it takes only a few minutes from the moment an IoT device is connected to the Internet to the first infection attempts. Still, we know little about the evolution of exploit vectors: Which vulnerabilities are being targeted in the wild, how has the functionality changed over time, and for how long are vulnerabilities being targeted? Understanding these questions can help in the secure development, and deployment of IoT networks.

We present the first longitudinal study of IoT malware exploits by analyzing 17,720 samples collected from three different sources from 2015 to 2020. Leveraging static and dynamic analysis, we extract exploits from these binaries to then analyze them along the following four dimensions: (1) evolution of infection vectors over the years, (2) exploit lifespan, vulnerability age, and the time-to-exploit of vulnerabilities, (3) functionality of exploits, and (4) targeted IoT devices and manufacturers. Our descriptive analysis uncovers several patterns: IoT malware keeps evolving, shifting from simply leveraging brute force attacks to including dozens of device-specific exploits. Once exploits are developed, they are rarely abandoned. The most recent binaries still target (very) old vulnerabilities. In some cases, new exploits are developed for a vulnerability that has been known for years. We find that the mean time-to-exploit after vulnerability disclosure is around 29 months, much longer than for malware targeting other environments.

ASIA CCS '22, May 30–June 3, 2022, Nagasaki, Japan

ACM ISBN 978-1-4503-9140-5/22/05.
https://doi.org/10.1145/3488932.3517408

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; **Vulnerability scanners**; • **Computer systems → Embedded systems**.

KEYWORDS

Static Analysis; Dynamic Analysis; Exploits; Vulnerabilities; Infection Vectors; Malware; IoT

ACM Reference Format:
Arwa Abdulkarim Al Alsadi, Kaichi Sameshima, Jakob Bleier, Katsunari Yoshioka, Martina Lindorfer, Michel van Eeten, and Carlos H. Gañán. 2022. No Spring Chicken: Quantifying the Lifespan of Exploits in IoT Malware Using Static and Dynamic Analysis. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30–June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3488932.3517408

1 INTRODUCTION

The widespread deployment of Internet-of-Things (IoT) devices, such as IP cameras and smart home appliances, bring us new services but also provides new resources for attackers to compromise. Indeed, devices have been infected at scale [4]. Attackers rely on exploits as infection vectors for IoT [3]—contrary to attacks on desktop and mobile devices, which have increasingly included vectors based on social engineering and user interaction.

While our knowledge has increased about IoT malware families and their capabilities [11, 61], we know much less about the attackers behavior in terms of the vulnerabilities they target with their exploit code. Just as the overall number of newly discovered vulnerabilities keeps growing [41], so does the number of IoT-related vulnerabilities: from a dozen or so reported in 2010 to more than 500 in 2019 [6]. Which of these vulnerabilities are targeted? Do the authors of different malware families go after the same vulnerabilities? How soon do they target a new vulnerability after it has been published? For how long do they keep focusing on a specific vulnerability? We have insights into these behaviors for desktops

and servers, where attackers predominantly target the last-but-one version of the software after a patch is released [1, 51, 58].

This pattern is unlikely to hold for IoT, however, since patching is more spotty and difficult in this ecosystem [55].

Earlier research has investigated exploit code used by specific malware families at specific times [4, 14, 24], but we lack a systematic understanding of how vulnerabilities are targeted over time across the IoT malware landscape. The closest related work is a concurrent study by Alrawi et al. [3]. The study conducted a wide-ranging analysis of IoT malware binaries collected in 2019 and identified 25 exploits observed in that year via static analysis. We end up confirming some of the results of this paper. We also go beyond it by directly observing the longitudinal evolution of exploits across five years and across malware families, as well as the timeline of the targeted vulnerabilities. We identify 63 exploits and track them over time, as well as the 68 vulnerabilities they are targeting. This allows us to present previously unobserved patterns in exploit lifetime, vulnerability lifetime, and time-to-exploit.

We base our analysis on IoT malware binaries from three different sources spanning five years (2015–2020). We analyzed 5,855 samples from a hybrid low-high interaction honeypot [43], collected between September 2018–August 2020, and 2,292 samples from the URLhaus malware repository [56], collected between July–October 2020. We clustered all binaries with AVClass2 [49] and Vhash [59] based on their binary similarity. For the URLhaus and honeypot datasets, we identified 156 clusters across 9 different malware families. We conducted dynamic analysis on both datasets to extract exploits. We also manually reverse engineered one sample for each cluster to extract further exploit functions. For all discovered exploits, we created a signature, which we could then trace further back in time using the Genealogy dataset, which was collected from VirusTotal [50] between 2015–2018 [13]. This allowed us to analyze the occurrence of these exploits in 17 additional malware families, beyond the 9 families from the URLhaus and honeypot datasets.

By mapping these exploits to vulnerabilities with and without published Common Vulnerabilities and Exposures (CVE) entries, we observe the evolution of infection vectors over the past five years. Whereas IoT malware started out exclusively focusing on brute-forcing credentials, infection vectors have since greatly diversified. Beyond brute forcing, we have identified 63 unique exploits across 26 families that target 68 vulnerabilities associated with 49 different manufacturers. The targeted vulnerabilities include recently discovered ones as well older ones, going back up to 12 years. Many vulnerabilities keep being targeted by new binaries for years on end, suggesting that patching is not rendering these attacks ineffective. In summary, our main contributions are:

- We present the first longitudinal study on IoT malware quantifying the exploits, targeted vulnerabilities and devices spanning 5 years. We find that attackers rarely abandon an exploit over time and that the number of targeted vulnerabilities has doubled every year since 2017.
- We create 63 unique signatures for the HTTP requests of each exploit and make these available to the community to facilitate detection.

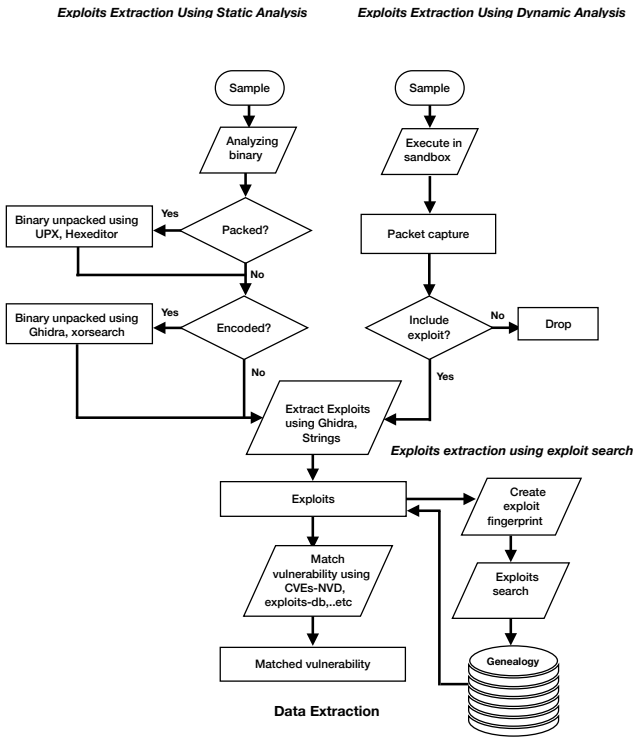


Figure 1: Analysis pipeline. We extract exploits using static and dynamic analysis from samples collected from 2018 to 2020 to create signatures, which we then match against a Genealogy dataset of unpacked samples from 2015 to 2018.

- The time between a vulnerability publication and the first occurrence of an exploit in a binary is 29 months on average, though this is skewed by some very old vulnerabilities.
- In terms of exploit lifespan, we see that on average exploits are used for 23 months, a number that is increasing as most exploits were still in use in the latest binaries.

We make the datasets, network artifacts, and developed signatures available at: https://doi.org/10.4121/19248725

2 METHODOLOGY

Our goal is to track the evolution of exploit code in IoT malware to gain insights into which vulnerabilities and devices they target over time. Identifying exploit code can be achieved via static and dynamic analysis of binaries. Automated static analysis is prone to code obfuscation and packing,

while manual static analysis—i.e., reverse engineering—is resource intensive, yet more complete and reliable. In contrast, automated dynamic analysis can deal with packing and is more scalable, but comes with the drawback of limitations in the coverage of exploits. We combine manual static analysis and automated dynamic analysis to leverage the strengths of each approach. We then supplement these results by a targeted exploit search in a binary repository that covers three years before our binaries were collected.

We present a high-level overview of our methodology in Figure 1.

Table 1: Number of collected samples per dataset, collection time period, and included malware families.

Dataset	# Binaries	# Shell Scripts	Malware Families (Avclass2)	# Vhash	# Singletons	Collection Period
URLhaus	2,292	6	Mirai, Gafgyt, Tsunami, Xorddos, Hajime, Mozi	108	6	Jul 2020 - Oct 2020
Honeypot	5,855	2,815	Mirai, Gafgyt, Tsunami, Xorddos, Hajime, Mozi, Berbew, Generica, Silex	107	59	Sep 2018 - Aug 2020
Genealogy	6,752	0	Mirai, Gafgyt, Tsunami, Hajime, Generica, Casur, Ddostf, Dnsamp, Dofloo, Gixec, Goram, Iotreaper, Luabot, Minerd, Pnscan, Qysly, Rakos, Ramgo, Remaiten, Satori, Shishiga and Ztorg	277	144	Jan 2015 - Aug 2018

2.1 Data Collection

2.1.1 IoT Malware Binaries. We collect current samples from two different sources (URLhaus and a honeypot) to create exploit signatures, which we then match against an older dataset (Genealogy) to cover an earlier period of time.

URLhaus: This repository collects URLs that are being used for distributing malware [56]. We used this source to collect a dataset of recent binaries as input for our dynamic and static analysis. From July 2020 to October 2020, we retrieved a daily file containing URLs of all captured binaries and other related information, such as file type. Since we focus on IoT malware, we extracted URLs only related to "Executable and Linkable Format" (ELF) files. We ran a daily script to download those files during the 4-month period. In total, we downloaded 2,298 binaries for different architectures including Renesas SH, Motorola 68000, SPARC, Intel 80386, ARM, PowerPC, MIPS, ARC Cores Tangent-A5, and AMD x86-64. Note that we include x86-based malware as previous work has shown that this is present in IoT devices [37].

While 6 of the 2,298 files were actually shell scripts, we manually verified that their functionality only included binary downloads and no propagation techniques.

Honeypot: We obtained 5,855 MIPS binaries from September 2018 to August 2020 via IoTPOT [43]. IoTPOT is a combination of low-interaction and high-interaction honeypot: the low-interaction honeypot emulates different network services, such as Telnet, HTTP front-ends, CPE WAN Management Protocol (CWMP), a backdoor of Netis routers, and the remote access setup service of several IP cameras. The high-interaction honeypot uses four bare-metal vulnerable IoT devices (a router, an IP camera, and two WiFi storage devices). The honeypot is connected to 130 IP addresses in Japan.

We also obtained a dataset of 2,815 files captured by IoTPOT whose file type were not ELF binaries. We executed them as shell scripts in a closed environment and found that 2,608 had the functionality of downloaders using wget, curl, etc. The remaining 207 files contained 10 Python scripts, two Perl scripts, and the remaining 195 files were recognized as ASCII texts but not script files. We then manually inspected these 10 Python and two Perl scripts and executed them in a sandbox and found that only one Python script includes exploits. Thus, we believe it is reasonable to focus the remainder of our analysis on the binary samples.

Genealogy: We supplemented the first two datasets with samples collected by a research project studying the genealogy of IoT malware [12, 13]. The dataset consists of IoT binaries submitted to VirusTotal [50] between January 2015 and August 2018. The authors selected binaries with file type *ELF* that had been detected by at least five anti-virus (AV) vendors, amounting to a total number of 93,652. For their study, they focus on a subset of 6,752 binaries, which they unpacked with their analysis pipeline. For our analysis we also use this set of unpacked binaries.

2.1.2 IoT Device Vulnerabilities. Once we identified exploits, we used the following sources to collect information about the targeted vulnerabilities:

National Vulnerability Database (NVD [35]): This database publishes Common Vulnerabilities and Exposures (CVEs) [54], including their identification number, a description, and public reference(s) for each vulnerability. Only 47 out of the 68 vulnerabilities (67.14%) that we identified have an associated CVE ID. This is a general issue also reported by VulnDB [60], which identified over 83,472 vulnerabilities that never received a CVE ID. We further found one exploit mapped to two CVEs from different years. We also encountered the opposite: two exploits targeting same CVE. Thus, an exploit can be linked to one or multiple vulnerabilities.

Exploit-DB [19]: We used this database of public exploit code and corresponding vulnerabilities as a source for exploits that were not assigned to any CVE.

Other public reports: For 15 exploits, there was no information in the above sources. We used keywords (e.g., strings found in HTTP headers, credentials, function names) to locate reports from AV vendors and researchers describing those vulnerabilities.

2.2 Malware Classification and Sampling

For all the binaries across the three datasets we used VirusTotal [50] to collect metadata about these files, such as the time a sample was first submitted to VirusTotal, its architecture, and AV labels. We further used AVClass2 [49] to assign malware family labels based on the majority of votes by the individual AV labels. Table 1 summarizes the results. If a sample does not have enough votes to assign it to a specific malware family, we label it as a Singleton [48].

Since manual reverse engineering is not feasible for thousands of binaries, we turned to VirusTotal in-house clustering based on Vhash [59]. It is a fuzzy hashing algorithm that clusters files based on their structural similarity.

In total, we obtained 108 different clusters for the URLhaus binaries and 107 clusters for the honeypot binaries. For the manual reverse engineering, we randomly selected one binary from each cluster. The binaries categorized as Singletons all have a unique Vhash, i.e., were assigned their own cluster. Therefore, we included all of them in our subset. Since binaries in each cluster are highly similar, we assume they also share the same exploit code. To double check this assumption, we randomly choose and reversed engineered two additional samples from five clusters. We found that all three binaries from each cluster indeed shared the same exploit code.

2.3 Exploit Extraction and Signature Matching

We perform dynamic analysis on the whole URLhaus and honeypot datasets, manual static analysis on a set of binaries sampled from

all clusters, and finally generate and match signatures against the Genealogy dataset.

2.3.1 Exploit Extraction using Dynamic Analysis. We executed each of the 2,298 binaries from URLhaus and 5,855 MIPS binaries from the honeypot in a malware sandbox. The sandbox is constructed as a virtual machine running Linux Debian for the MIPS, ARM, and x86-64 architectures using VirtualBox. we execute each binary for five minutes in an isolated network environment, except for DNS resolutions. This isolated environment helps us distinguish between port scans and C&C communications, especially when they are on the same port, as malware keeps trying to connect the C&C server in such an environment while port scans on the same host are not usually repeated. Moreover, we determine that a destination port is scanned if the number of destination IP addresses accessed by the binary on the port exceeds our heuristic threshold of 100. After the cleanup of the sandbox, we execute the binary again for five minutes with dummy servers running on the scanned ports. All connection attempts on the scanned port are then redirected to the dummy servers, implemented with PyNetSim, a network simulator for malware analysis. The current version of the dummy servers only establishes TCP sessions and does not respond any further. While this is a clear limitation, we find large portion of the binaries do start scanning and exploiting vulnerabilities right after their execution without interacting with C&C servers or their targets. During the dynamic analysis, 109 binaries from URLhaus and 4,700 binaries from the honeypot dataset were successfully executed and started exploiting.

We then examine the obtained attack payloads to check if they include any exploits. We manually extract the exploits as a characteristic sub-string of the payloads—for example, the target resource of HTTP GET and HTTP POST requests.

2.3.2 Exploit Extraction using Static Analysis. We conducted the manual static analysis on the subset of 156 binaries following the steps shown on the left side of Figure 1. The first step is to unpack the binary, if necessary. Malware authors use certain tactics to avoid detection and analysis. One of the most used methods is malware packing. A packer can compress, encrypt and obfuscate a program. Authors can also manipulate headers to corrupt them [31]. We run the binary in DIE [25] to check whether it is packed and, if so, by which packer. If the packer can be identified, we can use it to unpack the binary. For our samples, UPX [42] was the only packer we encountered. We could unpack 12 binaries with a simple `'upx -d filename'`. One sample from the Mozi family used custom UPX packing to hamper automatic unpacking. Customization to UPX can range from simply patching one byte, to more thorough modifications [32, 42]: modifying the UPX! magic headers, ELF magic bytes, the copyright string, section header names, and adding extra junk bytes throughout the binary are common. We unpack binaries packed with UPX manually by using a hexeditor [27] to fix the headers.

Next, we checked whether a binary is obfuscated or further encrypted by using `'strings -a'` [53] to check whether we can read the binary in plain text. If this is the case, we start then to read the whole binary and extract hard-coded credentials as well as exploits in HTTP requests. If this was not possible, we used Ghidra [39] to find the encryption key of 15 binaries and use one of its built-in scripts to decrypt them. If Ghidra could not locate the key, we identified the key ourselves and fed it to the built-in script for the decryption. We then either continued the analysis and exploits extraction in Ghidra or with `strings`. Some binaries still contained gibberish text even after the decryption. Therefore, we used XORSearch in those cases to make sure we extracted all the exploits. From our results from other binaries, we knew which keywords to use to help extract the exploits. For instance, we used `"http"`, `"GET /"`, `"POST /"` and other combinations from the HTTP syntax, as well as hard-coded credentials. If XORSearch found a decoded match, it returned the match with the XOR key used to encode it. Decoding the whole file by using this key allowed us investigate the binary further to find more hidden exploits.

2.3.3 Exploit Signature Generation and Matching. For every exploit we also created an exploit signature as an ordering of strings that can be used to uniquely identify an exploit. For each exploit in HTTP request format, we extracted the header. We used the request target, URI (Uniform Resource Identifier) or absolute path, following the GET or POST method in the request line. We used the standard URI syntax to generate signatures. The first box in Figure 2 shows the generic URI syntax that consists of a sequence of components: scheme, authority, path, query and fragment. The most common form is the origin form where an absolute path is followed by a '?' and then a query string. The bottom three boxes in Figure 2 show an example of how we generated an exploit signature. We extracted the HTTP request of an exploit and identified the URI syntax of it. Based on the information we found in public databases—e.g., Exploit DB—we confirmed the structure of the exploit signature. For example, in the third box, the parameter that comes after the path and the query was `'sourceUri'` that was triggering a command, then followed by the payload. The colored part might be adapted by attackers, so this could change the signature, while the remainder is stable.

While we mainly focus on exploits, we also looked for brute forcing capabilities as an infection vector. To this end, we used hard-coded credentials to create one signature for all binaries that rely on brute forcing. We used these signatures to extend the timeline of the exploits by tracing back their presence in the binaries from the Genealogy dataset. These binaries were collected between 2015–2018, so in the three years before the samples collected from URLhaus and the honeypot. We implemented a script to automate the exploit search based on matching the 63 exploit signatures besides binaries that rely on brute forcing credentials with the 6,752 unpacked binaries.

3 EXPLOIT LANDSCAPE

We now present the findings on exploits and vulnerabilities in IoT malware. Table 3 shows the findings across the three different datasets. Overall, we found 64 infection vectors: brute forcing hard-coded credentials, as well as 63 unique exploits that target 68 vulnerabilities. The occurrence of each vulnerability for each dataset is presented in the rightmost column in Table 3. We identify exploits, infection vectors, targeted vulnerabilities and device manufacturers.

Two groups of URLhaus binaries did not contain exploits and are not included in the table. The first group of 27 out of 108 (25%) binaries only included hard-coded credentials for brute forcing. A

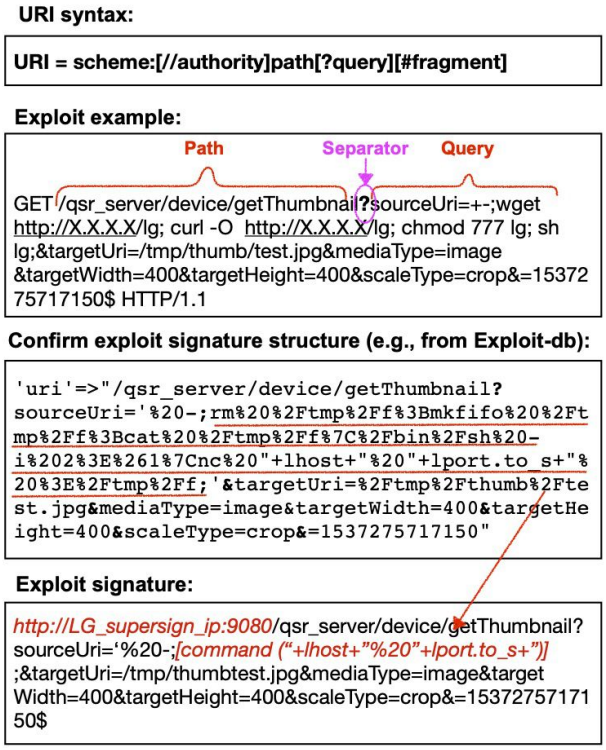


Figure 2: Example of a signature we generated for an exploit against CVE-2018-17173 [20, 40].

second group of 11 binaries (10%) did not include any infection vector in the code and only contained functions to receive commands from a command-and-control (C2) server and execute attacks. Prior work [29] has found that these commands implement attack vectors such as: UDP flood, SYN flood, ACK flood, TCP flood, UDP flood, VSE flood, DNS flood, GRE IP flood, GRE Ethernet flood and HTTP flood. These binaries belonged to Tsunami, XORDDoS, Hajime or were Singletons. For the remaining 70 binaries (65%), we found they contained 256 exploits targeting web vulnerabilities—i.e., exploits based on HTTP `GET /` and `POST /` requests.

In Table 3, we categorized the vulnerabilities into six groups, based on the vulnerability description in NVD or Exploit-DB: *Remote Code Execution (RCE)*, *Backdoor*, *Command Injection (CMDi)*, *Buffer Overflow*, *Web Application Firewall (WAF) Bypass* and *Brute Force*. More than half of the total vulnerabilities (55.88%) used Remote Code Execution (RCE) as the infection vector. RCE is also the most frequently targeted vulnerability type in both the URLhaus and the honeypot dataset, (55.9%) and (53.65%) respectively. Among the shared vulnerabilities across all the three datasets, CMDi was the most frequently used infection vector (56.25%).

Mirai used to infect vulnerable devices by using brute force, as was visible in its leaked source code [22] in 2016. More recent binaries have expanded the range of vectors: attackers now scan a number of different ports to exploit different protocols such as: Telnet, Android Debug Bridge (ADB), HTTP. Table 2 shows the

Table 2: Observation port scanning in the honeypot dataset during dynamic analysis.

Infection Vector	Protocol	Port
Backdoor	Telnet	443, 9530
Brute Force	Telnet	2223, 23, 23023, 2323, 2332, 26, 5885, 6000, 60000, 9001
Android Debug Bridge Shell CMDi	ADB	5555
Buffer Overflow	HTTP	443
Command Injection	HTTP	80, 88, 8081, 8080, 60001, 8000, 5501, 8082, 81, 8089, 8443, 5500, 49152, 443
Remote Code Execution	HTTP	37215, 1723, 55555, 443, 7547, 8080, 52869, 1024, 1234, 50000, 6666, 8001, 8081, 9080, 80, 8181, 9090, 161, 9000, 5555, 7574, 81

protocols and ports we observed during the dynamic analysis in respect to each infection vector.

The second column in Table 3 shows the 68 unique vulnerabilities. Those that share the same exploit code or are targeted by more than one exploit are marked with `'*'`. These are the vulnerabilities we extracted from the URLhaus and the honeypot. From the 69 infection vectors (including brute force), 31 are shared between both URLhaus and the Honeypot. The remaining 38 are 27 vulnerabilities found in only the URLhaus dataset and 11 vulnerabilities found in the Honeypot. When matching our exploit signatures against the older Genealogy dataset, we found 16 vulnerabilities matched by 17 exploits signatures. All of them were also present in the URLhaus (i.e., the most recent) dataset except for one: we only found the "SonicWall GMS-XMLRPC CMDi" in the Honeypot and the Genealogy datasets (see the last two columns in Table 3).

Not all vulnerabilities are recent ones: The third column in Table 3 shows the date each vulnerability was published. The oldest targeted vulnerability, CVE-2009-0545, was published in 2009 and dates back 12 years. However, the same exploit code was used to target another vulnerability, CVE-2019-12725, that was published ten years after that, in 2019. On the other hand, the most recent vulnerability, CVE-2020-17496, was published in August 2020 and exploited a month after that.

Mirai is the most versatile family: it targets all vulnerabilities except for three. This is consistent with the fact that Mirai contained the highest number of binary clusters, so it basically a family of families. The other malware families contained a subset of the exploits present in the Mirai family. For instance, in the URLhaus dataset, all of its vulnerabilities were present in Mirai samples except for one: CVE-2016-4429, which is a buffer overflow vulnerability in Qualcomm components that was only targeted by one binary categorized as a singleton (see rightmost column in Table 3).

Not only is Mirai the most versatile, it is also leading the way. Figure 3 shows a timeline of the occurrence of vulnerabilities and exploits in four malware families as well as singletons. Out of 68 vulnerabilities, 64 were observed first in Mirai samples before they were seen in other families. We discuss this figure in more detail in the next section. Mirai was also the only malware family with the most exploits per binary: up to 35 exploits, the maximum number

Table 3: Summary of infection vectors, vulnerabilities and devices across datasets (U=URLhaus, H=Honeypot, G=Genealogy). We identified 69 vectors (68 exploits + brute forcing default/weak credentials). ● indicates that we found samples exploiting a vulnerability across all three datasets, ◐ that we only found samples in two out of three datasets, and ○ that we only found samples in one dataset, demonstrating the value of samples from different vantage points and across time.

Type	Vulnerability	Vuln. Published	Exploit Published	Families	Manufacturer	Target Device	U	H	G	# of Samples
RCE	CVE-2009-0545; CVE-2019-12725 *	2009-02-12; 2019-06-04	2009-02-09	Mirai	Zeroshell	Zeroshell Linux Distribution	○			2
	Netgear DGN1000 RCE	2013-06-05	2013-06-05	Mirai, Mozi, Gafgyt	Netgear	DGN1000 Netgear routers	●	●	●	107
	Linksys E-series RCE	2013-07-02	2014-02-16	Mirai, Gafgyt	Cisco	Linksys routers E-series	◐	◐		150
	Edimax EW-7438RPn-v3 RCE	2015-07-17	2015-07-17	Mirai	Edimax	EW-7438RPn-v3	○			4
	Multi-vendor CCTV/DVR RCE	2016-03-23	2016-03-23	Mirai, Mozi, Gafgyt	Multi-vendor	Multi-vendor CCTV/DVR	●	●	●	79
	NUUO NVRmini RCE	2016-08-06	2016-08-06	Mirai	NUUO	NUUO NVR	◐	◐		4
	Xfinity Gateway RCE	2016-12-02	2016-12-02	Mirai	Xfinit	Xfinity Gateway	○			3
	CVE-2017-(8221-8225) *	2017-04-25	2017-03-08	Mirai	GoAhead	GoAhead IPcam	○			3
	EnGenius IoT GCS1.4.11 RCE	2017-06-04	2017-06-04	Mirai	EnGenius	EnGenius IoT Cloud Service	◐	◐		3
	CVE-2017-14135	2017-09-04	2017-07-03	Mirai	Dream Property	Opendreambox	○			1
	CVE-2017-14127; CVE-2019-18396 *	2017-09-04; 2019-10-24	2019-11-13	Mirai	Technicolor	Technicolor TD5336	◐	◐		5
	Vacron NVR RCE	2017-10-22	2017-10-08	Mirai, Mozi	Vacron	Vacron NVR devices	●	●	●	26
	Shenzhen_TVT RCE	2018-04-03	2018-04-09	Mirai	Shenzhen TVT	Shenzhen TVT DVR/NVR/IPC	○			3
	CVE-2018-10561; CVE-2018-10562 *	2018-04-30	2018-05-03	Mirai, Mozi, Gafgyt	Dasan	GPON Home Routers	●	●	●	259
	CVE-2018-11510	2018-05-28	2018-08-15	Mirai	ASUSTOR	ASUSTOR NAS		○		1
	HomeMatic Zentrale CCU2 RCE	2018-07-18	2018-07-18	Mirai	HomeMatic	HomeMatic Zentrale CCU2	○			3
	CVE-2018-15887	2018-08-26	2018-08-02	Gafgyt	ASUS	ASUS DSL-N12E_C1		○		6
	CVE-2018-17173	2018-09-18	2019-05-06	Mirai	LG	LG Supersign EZ CMS TV	◐	◐		9
	CVE-2018-20062; CVE-2019-9082 *	2018-12-11; 2019-02-24	2019-01-14; 2020-04-16	Mirai, Singletons	ThinkPHP	v-5.0.23/5.1.31 Server	◐	◐		21
	CVE-2019-2725	2018-12-14	2019-05-08	Mirai	Oracle	Oracle WebLogic Server		○		1
	CVE-2019-7276	2019-01-31	2019-11-12	Mirai	Optergy	Optergy 2.3.0a	○			3
	CVE-2019-10655	2019-03-30	2019-03-31	Mirai	Grandstream	GAC2500; GVC3202; GXV3275-40; GXP2200 *	○			3
	CVE-2018-20841	2019-06-11	2019-01-14	Mirai	HooToo	HT-TM05&HT-05 routers		○		1
	Sar2HTML 3.2.1 RCE	2019-08-02	2019-08-02	Mirai	Sar2HTML	Sar2html 3.2.1	○			3
	CVE-2020-9054	2020-02-18	2020-02-24	Mirai	Zyxel	Zyxel NAS and Firewall	◐	◐		6
	Netlink GPON Router 1.0.11 RCE	2020-03-18	2020-03-18	Mirai	Netlink GPON	Netlink GPON Router 1.0.11	○			58
	Symantec SWG 5.0.2.8 RCE	2020-04-09	2020-04-09	Mirai	Symantec	Symantec Web Gateway 5.0.2.8	◐	◐		34
	Netgear R7000 RCE	2020-06-15	2020-06-15	Mirai	Netgea	Netgear R7000	○			7
	CVE-2019-16759; CVE-2020-17496 *	2019-09-24; 2020-08-12	2020-08-12	Mirai	vBulletin 5.x	Servers using vBulletin 5.x	○			2
Backdoor	CVE-2014-2321	2014-03-10	2014-03-03	Tsunami	ZTE	ZTE F460 and F660		○		2
	Xiaongmai-based DVR/NVR/IPcam	2020-02-04	2020-02-04	Mirai, Gafgyt	Multi-vendor	DVR/NVR/IPcams		○		31
CMDi	CVE-2014-8361 *	2014-10-20	2015-06-01	Mirai, Mozi, Gafgyt	D-Link	D-Link Routers using Realtek SDK	●	●	●	272
	CVE-2014-9094	2014-11-26	2014-07-13	Mirai	WordPress	WordPress Plugin DZS-VideoGallery	◐		◐	35
	CVE-2015-2051	2015-02-23	2015-06-01	Mirai, Mozi, Gafgyt	D-Link	D-Link DIR-645	●	●	●	93
	AVTECH IPCam/NVR/DVR CMDi	2016-10-11	2016-10-11	Mirai	AVTECH	AVTECH IPcam/NVR/DVR	◐	◐		69
	CVE-2016-10372	2016-05-16	2016-11-08	Mirai, Mozi, Gafgyt	Zyxel	Eir D1000 Router (rebranded Zyxel)	◐	◐		78
	CVE-2016-6277	2016-07-22	2017-03-13	Mirai, Mozi, Gafgyt	Netgear	Netgear R7000 and R6400	●	●	●	41
	NUUO OS CMDi	2016-08-06	2016-08-06	Mirai	NUUO	NUUO NVRmini 2 3.0.8	○			3
	MV Power Shell CMDi	2017-02-27	2017-02-27	Mirai, Mozi	MV Power	MVPower DVR TV-7104HE 1.8.4	◐	◐		168
	CVE-2017-6884	2017-03-14	2017-04-02	Mirai	Zyxel	EMG2926 Router	◐		◐	39
	CVE-2017-18368	2019-05-02	2016-12-26	Mirai, Singletons, Gafgyt	Zyxel	Zyxel P660HN-T routers	◐	◐		77
	CVE-2017-17215	2017-12-04	2017-12-25	Mirai, Mozi, Gafgyt, Singletons	Huawei	Huawei home routers HG532	●	●	●	921
	CVE-2018-7841	2018-03-08	2019-05-14	Mirai	U.motion	U.motion software v.1.3.4	○			4
	D-Link DSL-2750B OS CMDi	2018-05-25	2018-05-25	Mirai	D-Link	D-Link DSL-2750B	●	●	●	241
	SonicWall GMS-XMLRPC CMDi	2018-08-01	2018-08-01	Mirai	SonicWall	SonicWall GMS		◐	◐	1
	CVE-2018-19276	2018-11-14	2019-12-18	Mirai	OpenMRS	OpenMRS before 2.24.0	◐	◐		5
	CVE-2019-7256	2019-01-31	2019-11-12	Mirai	Linear	Linear eMarge E3 series		○		1
	CVE-2019-12489	2019-05-30	2019-11-13	Mirai	Fastweb	Fastweb Fastgate 0.00.81	○			3
	CVE-2013-7471	2019-06-11	2013-09-17	Mirai, Mozi, Gafgyt	D-Link	D-Link DIR-645	●	●	●	29
	CVE-2019-14931	2019-08-10	2019-08-13	Mirai	Mitsubishi	Mitsubishi smartRTU& INEA ME-RTU	○			7
	CVE-2020-1956	2019-12-02	2020-06-20	Mirai	Apache	Apache Kylin 2.3.0-2.6.5,3.0.1	○			4
	CVE-2019-19824	2019-12-16	2015-07-16	Mirai	TOTOLINK	TOTOLINK Realtek SDK routers	○			7
	CVE-2020-5722	2020-01-06	2020-03-24	Mirai	Grandstream	Grandstream UCM6200 series	◐	◐		5
	CVE-2020-7209	2020-01-16	2020-05-17	Mirai	HP LinuxKI	HP LinuxKI-v6.01	○			3
	CVE-2020-10173	2020-03-05	2020-02-27	Mirai	Comtrend	Comtrend VR-3033	◐	◐		5
	CVE-2020-13786	2020-06-03	2020-06-12	Mirai	D-Link	D-Link DIR-865L Ax1.20B01	○			7
Buffer OF	CVE-2016-4429	2016-05-02	2016-05-18	Singletons	Qualcomm	Qualcomm Server	○			5
	CVE-2019-7405	2019-02-05	2019-12-16	Mirai	TP-Link	TP-Link Archer C5-v4 routers		○		4
WAF Bypass	Cloudflare WAF Bypass	2017-04-04	2016-10-25	Mirai, Gafgyt	CloudFlare	CloudFlare WAF	◐		◐	37
Brute Force	Dictionary Attack	-	-	Mirai, Mozi, Singletons, Tsunami, Gafgyt, xorddos	-	-	●	●	●	5,631
Total							59	41	16	

'*' indicates that this entry consists of vulnerabilities that are targeted by the same exploit code or vice versa

Table 4: Number of time device types were targeted by binaries, for each dataset, and in total.

Device Category	URLhaus	Honeypot	Genealogy	Total
Router	461	1,342	610	2,413
Home security	93	219	78	390
Web application	36	38	32	106
Web server	22	10	-	32
TV	7	2	-	9
NAS	27	27	-	54
Total	646	1,638	729	3,004

of exploits we observed in an single binary that was part of the most recent samples from the URLhaus dataset (see Figure 4).

The only vulnerability that is exploited by all IoT malware families and found across all the three datasets was CVE-2017-17215, which targets Huawei HG532 home routers. In terms of targeted manufacturers and devices, this vulnerability also had the highest frequency across binaries, which is (somewhat) visible in Figure 3 from the density of dots, and in last column in Table 3. Routers were the most favorable IoT device type for attackers to target over the period of six years. As can be seen in Table 4, 2,413 (80%) of targeted IoT devices were routers. In fact, this number is larger than summing up all the other IoT device types together.

4 EXPLOIT LIFESPAN

The number of IoT vulnerabilities, together with the number of exploits targeting these vulnerabilities, has been increasing over the years. Following the methodology explained in Section 2.3, we searched for matches for exploit signatures in the Genealogy dataset (2015–2018). Out of 64 exploit signatures, 17 signatures (associated with 16 vulnerabilities) had a match in the Genealogy dataset. Figure 3 illustrates one of the reasons for this limited presence of the exploits in older binaries: 32 (47%) of the vulnerabilities were published after August 2018, the end of the collection period of the Genealogy dataset. This still leaves 15 exploits targeting older vulnerabilities absent from the Genealogy dataset. Assuming that the Genealogy dataset is representative for that period, it would mean that these older vulnerabilities were selected by authors of more recent malware, years after they were first published. In total, we found matches in 5,421 samples out of the 6,752 binaries (80%). The lack of matches with the remaining 20% could reflect the use of abandoned exploit code that is no longer present in later binaries, but it is certainly also impacted by the number of packed and encoded samples in this dataset, as acknowledged by the researchers who created the repository. We discuss this limitation in Section 6.

With the extended timeline from the Genealogy dataset, we investigated the lifespan of the exploits, i.e., the time from when an exploit was first seen until the last time it was observed. We also investigate the *time-to-exploit*, i.e., the time that passes between the publication of the vulnerability and the first observation of a binary that contains exploit code for that vulnerability. For all binaries, we collected the 'first seen' date from VirusTotal.

Table 5: Number of hits (occurrence), exploits, and vulnerabilities per year.

Year	# Occurrences	# Exploits	# Vulnerabilities
2017	46	10	8
2018	727	15	15
2019	376	26	27
2020	1,855	58	63

Figure 3 visualizes the lifespan of malware exploit vectors by mapping the observations of the exploit in binaries (colored dots) and the publication dates of the underlying vulnerabilities (black X) and exploit code (red circle). In some cases, e.g., CVE-2013-7471, the date recorded in the CVE ID was years before the official publish date: 2019-06-11. We noticed in four more CVEs (CVE-2020-1956; CVE-2018-20841; CVE-2019-2725) that the publish date is not consistent with the CVE ID. That might explain why in some cases, the exploit publish date occurred well before the vulnerability publish date. The total number of exploits that were published before the official vulnerability disclosure date is 20, though in most cases the dates were relatively close together, so this order might reflect inaccuracies in the underlying data rather than the true order of events.

Figure 3 also presents the evolution of malware families.

All binaries that were first seen in 2015–2016 relied exclusively on hard-coded credentials for brute forcing. In total, these were 4,091 out of the 5,421 binaries. This might have been reinforced by the release of Mirai source code in November 2016 [22]. Mirai was the first IoT botnet able to amass millions of IoT devices and thus considered a leap in the IoT malware realm. Thus, since Mirai code had relied solely on brute forcing hard-coded credentials back then, the other variants and families appeared to follow suit. Since then, brute forcing has remained present in binaries up until the most recent data.

Figure 4 shows that, over time, the number of exploits per binary went up, as viewed by the difference across the datasets. Most binaries contain only one or two exploits, but from one dataset to the next, the upper part of the curve shifts towards the right. We found binaries in 2018 that had nine exploit signatures per individual sample. In binaries captured late 2020, as part of the URLhaus data, we found samples with up to 35 exploits, more than three times the number found in 2018. This is relative to the signatures we had, so it might undercount the exploits in the Genealogy dataset. Still, we see a similar increase compared to the honeypot binaries. Table 5 extends this picture by showing that not only individual binaries include more exploits, but that the total number of exploits rapidly increases over time across the malware landscape: from 2017 onwards, the total number of exploits and targeted vulnerabilities roughly doubles each year.

Even though the first exploits only emerged in 2017, four of the underlying vulnerabilities had already been published in 2013 or earlier (even before the start of the collection period of the Genealogy dataset). We calculated the time-to-exploit, which is the time between the vulnerability publication and the first time the associated exploit was observed in binaries. On average, the time-to-exploit is 29 months, though the underlying distribution is skewed

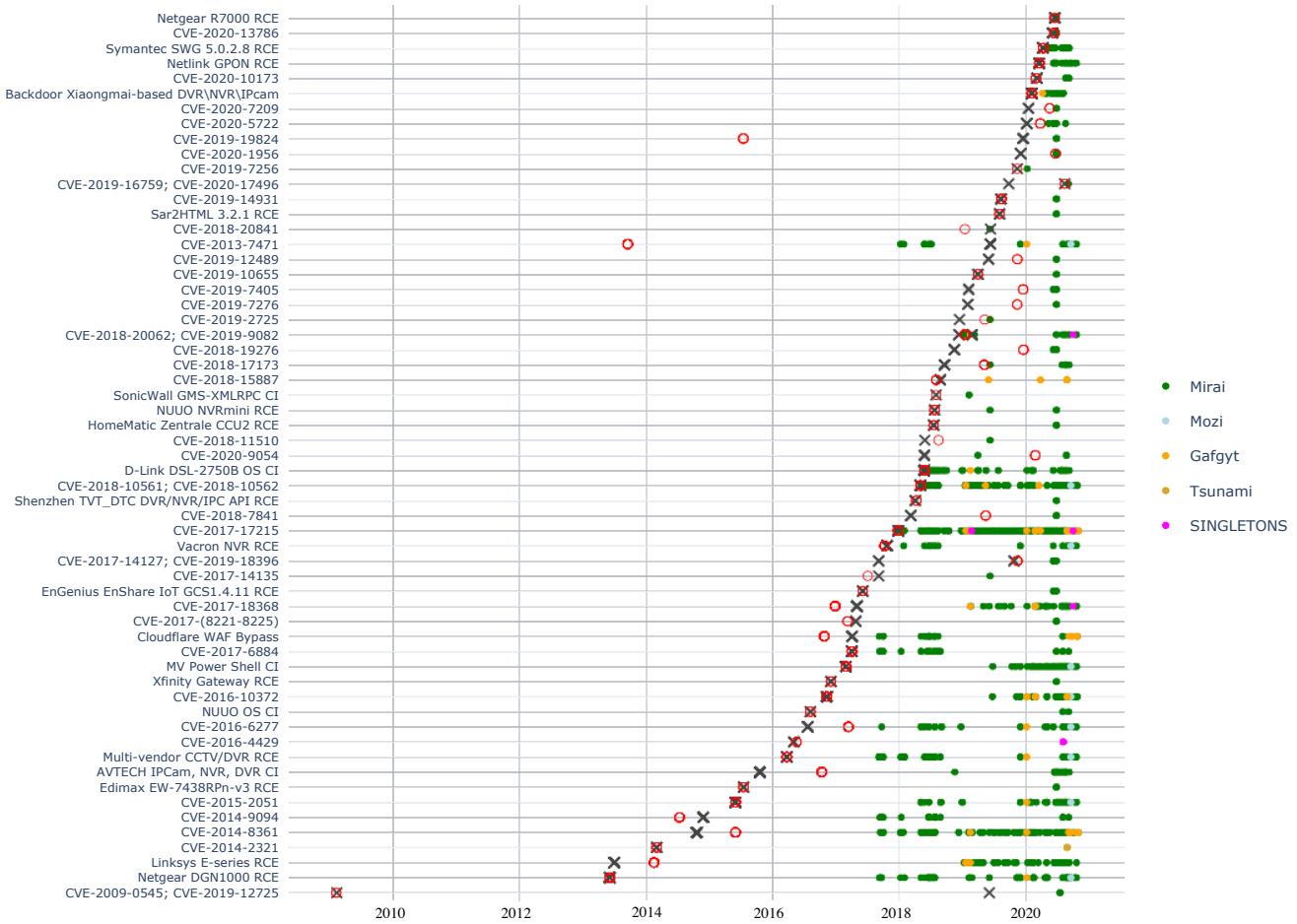


Figure 3: Occurrence of binaries that exploit vulnerabilities, per IoT malware family, ordered by vulnerability publication date. Each line represents a unique exploit. It might attack more than one vulnerability. The X symbol highlights the time the vulnerability was first published whereas the 0 symbol highlights the time the exploit was first discovered.

by very old vulnerabilities. The median is 26 months. When the authors of Mirai and other malware families started branching out from brute forcing to exploits, they apparently selected relatively old vulnerabilities to target first. This might reflect the device types they were interested in, namely routers and home security, as well as the size of install base of the targeted devices. Given that patching is spotty and difficult in the IoT ecosystem, they might have found that many devices were still vulnerable years after the vulnerability was first published.

For a closer inspection of the time-to-exploit for more recently discovered vulnerabilities, we calculated this metric for vulnerabilities published within the period of the honeypot binary collection (September 2018 – August 2020). All vulnerabilities published before September 2018 were excluded. This resulted in 326 exploit occurrences that mapped to 20 unique vulnerabilities (95% in Mirai and the remaining in Gafgyt). The mean time-to-exploit in 2019 was 40 days (median: 30 days). For 2020, when most exploits appeared, the mean was 135 days (median: 96 days). So there does not appear to be a decrease in the time-to-exploit in recent years. The overall average of 2019 and 2020 is 129 days.

The exploit lifespan is the period between when an exploit was first and last seen in binaries. Only in five cases did we observe a short attack window followed by a persistent absence of the exploit in future binaries. Most vulnerabilities are consistently targeted for months or years. On average, they are active for 23 months–and this number would be increasing, since these exploits were still present in the latest binaries we analyzed. Figure 5 also shows that the majority of vulnerabilities are exploited for months.

5 DISCUSSION

After Mirai's initial infection attempts by brute-forcing default (or weak) credentials, the arsenal of exploits has rapidly increased. Compared to prior work [3], which identified 25 exploits, a more alarming picture is emerging from our findings. Not only did we identify a larger set of exploits and targeted vulnerabilities, namely 68 of each, we also demonstrated that the evolution of exploits is gaining more and more velocity. Since 2017, the number of exploits and targeted vulnerabilities has roughly doubled every year.

Our findings also provide new insights into the attackers' behavior. For roughly half of all vulnerabilities, the attackers persist in their attacks for two years or more, while for the other half we see a pattern where an exploit is used for a short period and then abandoned. The latter might reflect a pattern of trial and error. If the exploit code is successful enough in recruiting bots, the attacks persist. The longer an attack persists, the more likely it is that the exploit code gets copied across clusters and families. Then, the vulnerability is consistently being attacked for years.

Another interesting finding is the selection of vulnerabilities by the attackers. IoT malware developers select old vulnerabilities, compared to malware developers for desktop operating systems or server software. The latter have been found to focus on reverse engineering the latest vulnerabilities from patch releases and to then attack the one-but-last version of the software [58]. For desktop and server software, the time-to-exploit—i.e., the time between the publication of a vulnerability and the first observation of a binary attacking that vulnerability—typically ranges from a single day (e.g., "Exploit Wednesday" following on Microsoft's "Patch Tuesday") to, at most, a few months for a few high profile attacks as Wannacry and Not-Petya [18, 63]. This strategy makes sense, given that the largest share of vulnerable systems is running the one-but-last version of the software.

We see a dramatically different pattern for IoT malware. The mean time-to-exploit is a stunning 29 months (median is 26 months). This attacker strategy is rational if devices are not really updating their software, since then whatever version was running on the devices when they were originally deployed into the market will remain the same. While new releases might or might not be developed by the manufacturer, our evidence clearly suggests that for the targeted devices, available patches are not being adopted by the device owners. In fact, a recent study by Xie et al. [64] found that, on average, 64.3% of unpatched devices were affected by 176 vulnerabilities due to lack of updates. A recent illustration of this attacker strategy was provided by the so-called Meris botnet, which launched "record-shattering" DDoS attacks against Yandex and Cloudflare [28]. The botnet consisted of 250,000 compromised MikroTik routers that were running different versions of the RouterOS software. By far the largest share was running 6.45.9 [45], which is a whopping 21 versions behind the latest stable release.

How can we improve our defense against the increasing velocity of IoT exploit development? Our data suggests that core of the issue is patching—or rather, the lack of patching. Assuming the observed attacker behavior is rational, then the owners of the devices are not installing updates for years or perhaps ever. This is consistent with other research [57]. The obvious path forward would then be to educate users on the importance of patching these devices.

The problem with relying on users to patch their devices is that manufacturers do not always make patches available and, even if they do, the install path is often far from user-friendly. Many users have incorrect mental models for IoT malware infections. They might see the devices as appliances rather than computers [17, 47]. One study on IoT malware remediation found that many users had no clue on how to interact with their device, even for relatively simple tasks like changing the default password [7].

Another key stakeholder for mitigating this problem is the population of Internet Service Providers (ISPs). The bulk of all IoT

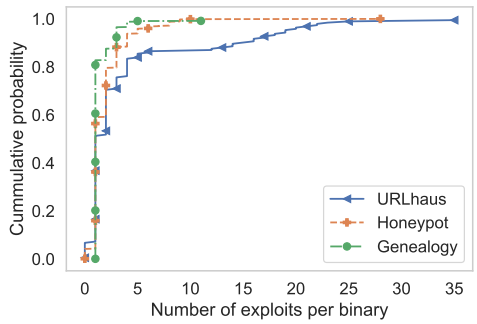


Figure 4: CDF of the number of exploits per sample.

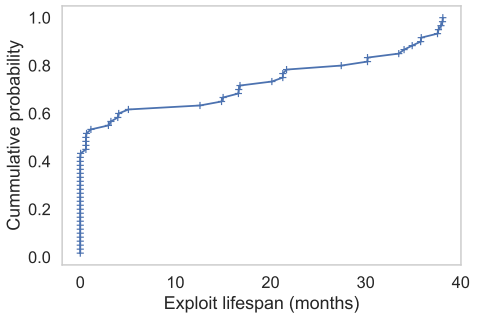


Figure 5: CDF of lifespan of each exploit.

infection reside in the networks of consumer ISPs [8, 9, 38, 46]. Increasingly, ISPs are taking on the task of notifying customers with infected IoT devices, in line with best practices for mitigating malware on personal computers. While remediating malware on, say, Windows machines has become much more user-friendly and effective over the year, the situation for IoT is totally different. There are tens of thousands of different devices in the market [30, 46], all with different remediation paths. Unable to handle this enormous diversity, ISPs and governments have been resigned to providing users with very generic remediation advice that is hardly actionable. Our findings suggest a different way forward: while more and more vulnerabilities are being targeted, the total set is actually finite and somewhat manageable in size: 68 in all. ISPs could be provided with remediation advice for each of these vulnerabilities and their corresponding devices and then point their infected customers to this finite set to see if they have a device on this list.

While it will be hard for manufacturers to repair these intrinsic problems with the devices that are already deployed in the market, our findings do underline the critical importance of user-friendly update mechanisms for new devices. A related issue is the duration of manufacturer support. We observed that attackers selected vulnerabilities that are many years old, in one case even a decade old. No manufacturer is currently supporting IoT devices on those time scales. This plays to the attackers advantage. They now have very attractive proposition: spend some effort on exploit development for an easy-to-exploit vulnerability on a device with a large install base and then recoup this investment by exploiting it for many

years, since patching is not happening fast enough and at some point will stop altogether, while the devices remain in the field.

6 LIMITATIONS AND FUTURE WORK

We rely on a number of external sources and tools, and thus inherit their limitations. First, during the data collection, we found that 17 of the exploits had no CVE ID, which makes it harder to accurately measure the disclosure date and lifespan of a vulnerability. Even for published CVEs, these fields have idiosyncrasies, where publication might not happen until years after the CVE process has been started, as indicated by the year in the CVE's ID.

Second, there is some ambiguity in the malware family classification. We followed best practices by using AVClass2 to normalize labels, but we could not assign 209 of the samples to a specific family and had to treat them as singletons. For further clustering to select samples for our manual analysis, we relied on light-weight fuzzy hashing. Telfhash [33] is widely used by malware analysts, and Pagani et al. [44] showed that it is robust to minor source code changes. In our experiments, Telfhash [33] failed to cluster 48% of samples, and had issues with packed files and corrupted headers. This indicates that Telfhash potentially puts too much emphasis on features from the ELF headers, making it less suitable in our scenario. We achieved better results with the functional similarity as calculated by VirusTotal's Vhash, which unfortunately is not publicly documented. To gain insights into the quality of our classification and clustering based on AVClass2 and Vhash, we compared it against a binary similarity metric using BinDiff [65] on the packed and unpacked URLhaus dataset. The results can be found in Appendix A. Although there are differences, overall there is enough consistency among the clusters identified by BinDiff and those of Vhash (as marked by the different colors) to provide confidence that Vhash is in line with state-of-the-art tools. In general, comparing whole binaries can be very susceptible to obfuscation and it might be better to search for similar functionality instead (e.g. [34]).

Our analysis pipeline faces the same limitations as other studies based on static and dynamic analyses techniques in a malware context: malware writers tend to include anti-analysis and obfuscation techniques to evade detection. Especially code packing is an issue for static analysis, and therefore not only an issue for our clustering, but also for our static exploit signature matching.

During dynamic analysis we faced the issue of samples not executing correctly, as also reported by Alrawi et al. [3]. We investigated this issue by running the same binaries using two different sandbox configurations (vm-sandbox and docker-sandbox). We found that both the number of binaries and amount of traffic captured was 84% higher in the former. Still, improving dynamic analysis coverage is an open and orthogonal research problem.

In terms of exploit coverage, we consider privilege escalation (i.e., exploits after the initial infection) out of scope. Extending our analysis to capture these exploits as well should be straightforward, especially as we expect the Yara signatures created by Alrawi et al. [3] as part of concurrent work in this area can be transformed into our signature format with little effort (and vice versa).

Finally, in terms of malware coverage, it is hard to estimate how representative our dataset is for IoT malware. We made our best effort to collect a dataset that is as diverse as possible. We collected samples from different vantage points, both actively through a honeypot and passively from public repositories and other studies.

7 RELATED WORK

IoT security research has traditionally focused on designing appropriate security controls for resource-constrained devices, yet few studies have looked into the security of already deployed IoT devices and their vulnerabilities. Feng et al. [21] studied IoT vulnerabilities using different sources in the wild such as public vulnerability and exploit databases, forums, mailing lists and blogs to propose using them for more effective defenses. Using similar data sources but leveraging machine learning, Blinowski and Piotrowski [6] proposed a vulnerability classification based on the CVE of IoT systems. Putting aside open data, Alrawi et al. [2] analyzed the vulnerabilities of a subset of home-Based IoT deployments in the first empirical evaluation of the security controls and vulnerabilities present in IoT devices already in the market.

While these previous studies focused on the defensive side of IoT security, recent research also looked into attacks by analyzing IoT malware [3, 5, 10, 14, 16]. These studies either leverage honeypots to capture IoT malware (e.g., IoTPOT [43]), collect samples from VirusTotal [50], or use open threat intelligence data (e.g., CyberIOCs [15]). For instance, Hamou-Lhadj and Razgallah [23] investigated which CVEs are more likely to be targeted by IoT malware based on public reports. They showed that IoT malware targets vulnerabilities that can be exploited remotely and do not require user interaction to compromise the device.

Recently, Alrawi et al. [3] analyzed a set of 166,000 IoT malware samples collected in 2019 with the aim of understanding code reuse and evolution of different IoT malware families. Similar to our research methodology, the authors used both static and dynamic analyses to identify similarities and differences across the different malware samples. Our work extends their initial effort in four ways that allow us to characterize different types of exploits and their evolution during a larger time period: (1) we provide a much more comprehensive understanding of the targeted vulnerabilities analyzing 68 vulnerabilities (excluding hard-coded credentials) as found in the binaries, to just 25 studied by Alrawi et al. [3], by analyzing 17,720 binaries from three types of data sources; (2) we covered binaries from a much larger time frame, 2015 to 2020, while Alrawi et al. [3] analyzed only one year of binaries from 2019; (3) we extracted exploits via a combination of static analysis, dynamic analysis, and signature matching, rather than only static analysis and YARA signatures; and (4) we longitudinally measured exploit lifetime, vulnerability lifetime, and time-to-exploit, which Alrawi et al. [3] did not measure at all. For the exploits that they discovered, they only identified a single point in time when the first industry report mentioned the exploit. This is not observed from their own data, nor is it measured longitudinally–e.g., the time frame in which binaries use the exploit. Thus, our work presents the first stepping stone to understand how attackers regularly target (very) old vulnerabilities and how they persist in certain attacks for years.

Focusing in vulnerabilities in general, several authors have analyzed the life cycle of vulnerabilities. For instance, Spanos and Angelis [52] argued that vulnerabilities' descriptions can be used to predict vulnerability characteristics. They developed a model

that combined analyzing texts and multi-target classification to determine the characteristics, severity, impact and score of a vulnerability. Analogously, Wijayasekara et al. [62] developed a text mining classifier to identify hidden impact vulnerabilities, i.e., vulnerabilities identified only long after their bugs report released in public databases. They found out that hidden impact vulnerabilities has increased from 25% to 36% in Linux and from 59% to 65% in MySQL within two years.

Finally, some authors also looked at the relationship between vulnerabilities and exploits. Nayak et al. [36] found that 85% of known vulnerabilities are never exploited in the wild. They introduced different security metrics such as the count of vulnerabilities exploited and the size of the attack surface. They conducted an empirical study of security in the deployment environment using around 300 million reports of intrusion-protection telemetry that were collected from more than six million hosts. They found no single product within their study that had more than 35% of their disclosed vulnerabilities exploited in the wild. Also, newer products or newer product versions tend to decrease the exploitation ratio and the exercised attack surfaces. Householder et al. [26] focused on answering 'when and how many vulnerabilities get associated public exploits' via analyzing CVE-IDs to find out how they influence exploit publication. They found that around 4% of the published CVEs get a public exploit code associated with them within 365 days. They argued that the exploit publication likelihood increase is influenced by CVSS score, CWE, and the recent mechanism of publishing CVE-ID. They studied 75,807 vulnerabilities for which they found that only 3,164 of them had public exploits during the whole 6 years of study. Those exploits have 2 days as a median time to publication whereas the mean time is 91 days.

8 CONCLUSION

In this paper, we performed the first longitudinal measurement study leveraging multiple vantage point to analyze the IoT malware ecosystem and underlying dynamics. Using static and dynamic analysis, as well as signature matching, we extracted the 63 unique exploits from 17,720 binaries belonging to 26 different IoT malware families. Our results show the ecosystem has diversified, from generic brute force attacks to embody a wide variety of device-specific exploits.

Mirai is the family that has evolved the most since its inception in 2016 and it still is the leading innovator. Most exploits were observed in Mirai first. Other malware families followed the same trend increasing the complexity of IoT malware and at the same time targeting more IoT devices and different protocols. The landscape is rapidly evolving: the number of exploits and targeted vulnerabilities has doubled every year since 2017.

Once exploits are developed, they are rarely abandoned. Many still appear in the most recent binaries. In our case, the exploit lifespan is longer than 5 years, though the duration of exploits is 38 months, on average.

Attackers target (very) old vulnerabilities. The mean time-to-exploit between the publication of the vulnerability and the first occurrence of an exploit in a binary is 29 months on average, though this time frame varies wildly across exploits. This is very different from the patterns we observe for malware targeting desktop and server software. Assuming this different attacker strategy for IoT is rational, then our evidence suggests that the targeted IoT devices are rarely, if ever, patched. Thus, windows for exploiting a vulnerability do not decrease rapidly over time. The age of the vulnerability is much less relevant to attackers than the size of the install base of the device and the easy with which exploit vectors can be developed. Once they are developed, they remain in use for years.

Our study clearly shows that attackers are taking advantage of certain weaknesses in the IoT ecosystem, most notably the lack of patching and the diversity of devices and manufacturers–recently estimated to consist of over 14,000 different companies [30]. This is a target-rich environment where device not only have different vulnerabilities, but also their own paths and dead-ends towards making them more secure against malware. We identified a number of implications of our findings for users, Internet Service Providers and manufacturers.

ACKNOWLEDGMENTS

This work is partly supported by the Dutch Research Council (NWO) under the RAPID project (Grant No. CS.007) and the "Hestia Research Programme" (Grant No. VidW.1154.19.011). This research is partly funded by King Abdulaziz City for Science and Technology (KACST). A part of this research was conducted in "MITIGATE" project among "Research and Development for Expansion of Radio Wave Resources (JPJ000254)", supported by the Ministry of Internal Affairs and Communications, Japan. The research has further received funding from the Vienna Science and Technology Fund (WWTF) through project ICT19-056 (IoTIO), and SBA Research (SBA-K1), a COMET Centre within the framework of COMET – Competence Centers for Excellent Technologies Programme and funded by BMK, BMDW, and the federal state of Vienna. The COMET Programme is managed by FFG.

REFERENCES

[1] Luca Allodi. 2017. Economic Factors of Vulnerability Trade and Exploitation. In *Proceedings of the ACM SIGSAC Conf. on Computer and Communications Security.*

[2] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security Evaluation of Home-Based IoT Deployments. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P).* https://doi.org/10.1109/SP.2019.00013

[3] Omar Alrawi, Charles Lever, Kevin Valakuzhy, Ryan Court, Kevin Snow, Fabian Monrose, and Manos Antonakakis. 2021. The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle. In *30th USENIX Security Symposium (USENIX Security 21).* USENIX Association, 3505–3522.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the Mirai Botnet. In *Proceedings of the USENIX Security Symposium.* 1093–1110.

[5] Afsah Anwar, Jinchun Choi, Abdulrahman Alabduljabbar, Hisham Alasmary, Jeffrey Spaulding, An Wang, Songqing Chen, DaeHun Nyang, Amro Awad, and David Mohaisen. 2021. Understanding Internet of Things Malware by Analyzing Endpoints in their Static Artifacts. *arXiv preprint arXiv:2103.14217* (2021).

[6] Grzegorz J Blinowski and Paweł Piotrowski. 2020. CVE Based Classification of Vulnerable IoT Systems. In *Proceedings of the International Conference on Dependability and Complex Systems (DepCoS).* 82–93.

[7] Brennen Bouwmeester, Elsa Rodríguez, Carlos Gañán, Michel van Eeten, and Simon Parkin. 2021. "The Thing Doesn't Have a Name": Learning from Emergent Real-World Interventions in Smart Home Security. In *Proceedings of the USENIX Symposium on Usable Privacy and Security (SOUPS).* USENIX Association.

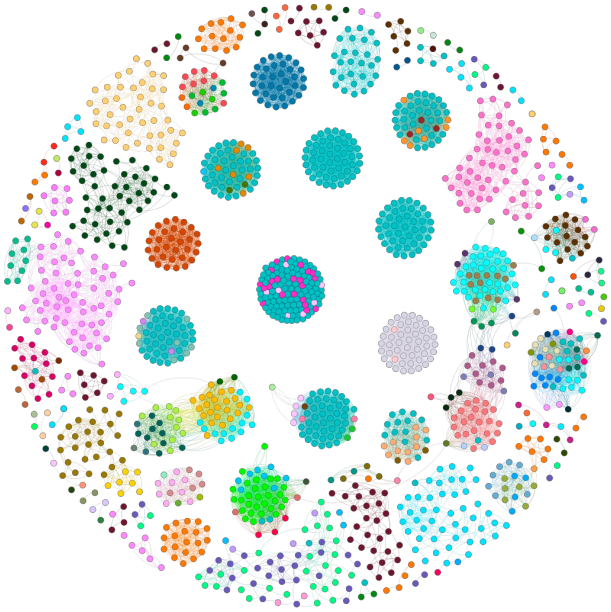
[8] Orçun Çetin, Carlos Gañán, Lisette Altena, Takahiro Kasama, Daisuke Inoue, Kazuki Tamiya, Ying Tie, Katsunari Yoshioka, and Michel van Eeten. 2019. Cleaning Up the Internet of Evil Things: Real-World Evidence on ISP and Consumer Efforts to Remove Mirai. (2019). https://doi.org/10.14722/ndss.2019.23438

[9] Orçun Çetin, Carlos Ganán, Lisette Altena, Samaneh Tajalizadehkhoob, and Michel Van Eeten. 2019. Tell Me You Fixed It: Evaluating Vulnerability Notifications via Quarantine Networks. In *2019 IEEE European Symposium on Security*

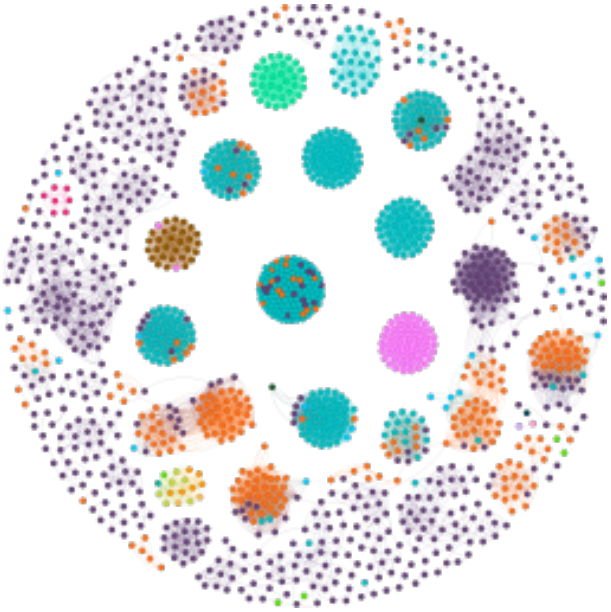
and Privacy (EuroS&P). IEEE, 326–339.
[10] Jinchun Choi, Afsah Anwar, Hisham Alasmary, Jeffrey Spaulding, DaeHun Nyang, and Aziz Mohaisen. 2019. IoT Malware Ecosystem in the Wild: A Glimpse into Analysis and Exposures. In *Proceedings of the ACM/IEEE Symposium on Edge Computing (SEC)*. 413–418.
[11] Andrei Costin and Jonas Zaddach. 2018. IoT Malware: Comprehensive Survey, Analysis Framework and Case Studies. *BlackHat USA* (2018).
[12] Emanuele Cozzi. 2021. The Tangled Genealogy of IoT Malware Dataset. https://github.com/eurecom-s3/tangled_iot/tree/master/dataset
[13] Emanuele Cozzi, Sophia Antipolis, France Pierre-Antoine Vervier France Matteo Dell, France Yun Shen, Leyla Bilge, Davide Balzarotti, Pierre-Antoine Vervier, Matteo Dell, and Yun Shen. 2020. The Tangled Genealogy of IoT Malware. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.
[14] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. 2018. Understanding Linux Malware. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, Vol. 2018-May. 161–175.
[15] CyberIOCs. 2020. CyberIOCs Daily malware pack. https://freeiocs.cyberiocs.pro
[16] Fan Dang, Zhenhua Li, Yunhao Liu, Ennan Zhai, Qi Alfred Chen, Tianyin Xu, Yan Chen, and Jingyu Yang. 2019. Understanding Fileless Attacks on Linux-based IoT Devices with Honeycloud. In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 482–493.
[17] Antoine d'Estalenx and Carlos Gañán. 2021. NURSE: eNd-UseR IoT malware detection tool for Smart homEs. In *Proceedings of the 11th International Conference on the Internet of Things (IoT '21)*. 1–8.
[18] Michel Edkrantz, Staffan Truvé, and Alan Said. 2015. Predicting Vulnerability Exploits in the Wild. In *Proceedings on the IEEE International Conference on Cyber Security and Cloud Computing*. 513–514.
[19] Exploit-db. 2009. *Exploit Database - Exploits for Penetration Testers, Researchers, and Ethical Hackers*. Retrieved June 15, 2021 from https://www.exploit-db.com/
[20] Alejandro Fanjul. 2018. *LG SuperSign EZ CMS 2.5*. Retrieved May 01, 2021 from https://www.exploit-db.com/exploits/45448
[21] Xuan Feng, Xiao Liao, X Wang, Qiang Li, Kai Yang, Hong Zhu, and Limin Sun. 2019. Understanding and Securing Device Vulnerabilities through Automated Bug Report Analysis. In *Proceedings of the USENIX Security Symposium*.
[22] Jerry Gamblin. 2016. Mirai Source Code. https://github.com/jgamblin/Mirai-Source-Code
[23] Abdelwahab Hamou-Lhadj and Asma Razgallah. 2021. An Analysis of the Use of CVEs by IoT Malware. In *Proceedings of the International Symposium on Foundations and Practice of Security (FPS)*, Vol. 12637. 47.
[24] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. 2019. Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet.
[25] Hors. 2021. Detect-It-Easy: Program for determining types of files for Windows, Linux and MacOS. https://github.com/horsicq/Detect-It-Easy
[26] Allen D Householder, Jeff Chrabaszcz, Trent Novelly, David Warren, and Jonathan M Spring. 2020. Historical Analysis of Exploit Availability Timelines. In *Proceedings of the USENIX Workshop on Cyber Security Experimentation and Test*.
[27] Maël Hörz. 2020. HxD - Freeware Hex Editor and Disk Editor | mh-nexus. https://mh-nexus.de/en/hxd/
[28] Brian Krebs. 2021. KrebsOnSecurity Hit By Huge New IoT Botnet "Meris". https://krebsonsecurity.com/2021/09/krebsonsecurity-hit-by-huge-new-iot-botnet-meris/
[29] Ayush Kumar and Teng Joon Lim. 2018. A Secure Contained Testbed For Analyzing IoT Botnets. In *Proceedings of the International Conference on Testbeds and Research Infrastructures (TridentCom)*. 124–137.
[30] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. 2019. All Things Considered: An Analysis of IoT Devices on Home Networks. In *Proceedings of the USENIX Security Symposium*. USENIX Association, Santa Clara, CA, 1169–1185.
[31] Keane Lucas, Mahmood Sharif, Lujo Bauer, Michael K. Reiter, and Saurabh Shintre. 2021. Malware Makeover: Breaking ML-based Static Analysis by Modifying Executable Bytes. In *Proceedings of the ACM ASIA Conference on Computer and Communications Security (AsiaCCS)*.
[32] Malwaremustdie. 2018. Unpacking the non-unpackable. https://github.com/radareorg/r2con2018/blob/master/talks/unpacking/Unpacking-a-Non-Unpackables.pdf
[33] Fernando Mercês and Joey Costoya. 2020. *Telfhash: An Algorithm That Finds Similar Malicious ELF Files Used in Linux IoT Malware*. Technical Report.
[34] Azqa Nadeem, Christian Hammerschmidt, Carlos H Gañán, and Sicco Verwer. 2021. Beyond labeling: Using clustering to build network behavioral profiles of malware families. In *Malware Analysis Using Artificial Intelligence and Deep Learning*. Springer, Cham, 381–409.
[35] National Institute of Standards and Technology (NIST). 2021. National Vulnerability Database. https://nvd.nist.gov/ (Accessed on 2021-06-16).
[36] Kartik Nayak, Daniel Marino, Petros Efstathopoulos, and Tudor Dumitraş. 2014. Some Vulnerabilities are Different than Others: Studying Vulnerabilities and Attack Surfaces in the Wild. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*.
[37] Weina Niu, Xiaosong Zhang, Xiaojiang Du, Teng Hu, Xin Xie, and Nadra Guizani. 2019. Detecting Malware on x86-based IoT Devices in Autonomous Driving. *IEEE Wireless Communications* 26, 4 (2019), 80–87.
[38] Arman Noroozian, Elsa Turcios Rodriguez, Elmer Lastdrager, Takahiro Kasama, Michel Van Eeten, and Carlos Gañán. 2021. Can ISPs Help Mitigate IoT Malware? A Longitudinal Study of Broadband ISP Security Efforts. In *IEEE European Symposium on Security and Privacy*.
[39] NSA. 2020. Ghidra. https://ghidra-sre.org/
[40] NVD. 2018. CVE-2018-17173. https://nvd.nist.gov/vuln/detail/CVE-2018-17173
[41] NVD. 2021. CVSS Severity Distribution Over Time. https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time
[42] Markus F.X.J. Oberhumer, László Molnár, and John F. Reiser. 2020. UPX - the Ultimate Packer for eXecutables. https://github.com/upx/upx
[43] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. 2015. IoTPOT: Analysing the Rise of IoT Compromises. In *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*.
[44] Fabio Pagani, Matteo Dell'Amico, and Davide Balzarotti. 2018. Beyond Precision and Recall: Understanding Uses (and Misuses) of Similarity Hashes in Binary Analysis. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*.
[45] Qrator. 2021. Mēris botnet, climbing to the record. https://blog.qrator.net/en/meris-botnet-climbing-to-the-record_142/
[46] Elsa Rodríguez, Arman Noroozian, Michel van Eeten, and Carlos Gañán. 2021. Superspreaders: Quantifying the Role of IoT Manufacturers in Device Infections. *Workshop on the Economics of Information Security (WEIS)* (2021).
[47] Elsa Rodríguez, Susanne Verstegen, Arman Noroozian, Daisuke Inoue, Takahiro Kasama, Michel van Eeten, and Carlos Gañán. 2021. User compliance and remediation success after IoT malware notifications. *Journal of Cybersecurity* 7, 1 (2021).
[48] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. 230–253.
[49] Silvia Sebastián and Juan Caballero. 2020. AVclass2: Massive Malware Tag Extraction from AV Labels. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. 42–53.
[50] Gaurav Sood. 2021. VirusTotal. https://www.virustotal.com/gui/home/search
[51] Kyle Soska and Nicolas Christin. 2014. Automatically Detecting Vulnerable Websites Before They Turn Malicious. In *Proceedings of the USENIX Security Symposium*. USENIX Association, San Diego, CA, 625–640.
[52] Georgios Spanos and Lefteris Angelis. 2018. A Multi-target Approach to Estimate Software Vulnerability Characteristics and Severity Scores. *Journal of Systems and Software* (2018).
[53] Strings. 2009. *Strings(1) - Linux man page*. Retrieved November 1, 2020 from https://linux.die.net/man/1/strings
[54] The MITRE Corporation. 2021. CVE - Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/index.html
[55] Hannes Tschofenig and Stephen Farrell. 2017. Report from the Internet of Things Software Update (IoTSU) Workshop 2016. RFC 8240.
[56] URLhaus. [n. d.]. *URLhaus | Malware URL exchange*. https://urlhaus.abuse.ch/
[57] Kami Vaniea, Emilee J. Rader, and Rick Wash. 2014. Betrayed by updates: how negative experiences affect future security. In *CHI Conference on Human Factors in Computing Systems*, Matt Jones, Philippe A. Palanque, Albrecht Schmidt, and Tovi Grossman (Eds.). ACM, 2671–2674.
[58] Marie Vasek and Tyler Moore. 2014. Identifying Risk Factors for Webserver Compromise. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)*, Nicolas Christin and Reihaneh Safavi-Naini (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 326–345.
[59] VirusTotal. 2020. VirusTotal Reference API | Files. https://developers.virustotal.com/v3.0/reference#files
[60] Vulndb. 2020. Comprehensive Vulnerability Intelligence. https://vulndb.cyberriskanalytics.com/#statistic (Accessed on 2021-05-01).
[61] Huanran Wang, Weizhe Zhang, Hui He, Peng Liu, Daniel Xiapu Luo, Yang Liu, Jiawei Jiang, Yan Li, Xing Zhang, Wenmao Liu, Runzi Zhang, and Xing Lan. 2021. An Evolutionary Study of IoT Malware. *IEEE Internet of Things Journal* (2021).
[62] Dumidu Wijayasekara, Milos Manic, Jason Wright, and Miles McQueen. 2012. Mining Bug Databases for Unidentified Software Vulnerabilities. In *Proceedings of the International Conference on Human System Interactions (HSI '12)*. 89–96.
[63] Chaowei Xiao, Armin Sarabi, Yang Liu, Bo Li, Mingyan Liu, and Tudor Dumitras. 2018. From Patching Delays to Infection Symptoms: Using Risk Profiles for an Early Discovery of Vulnerabilities Exploited in the Wild. In *Proceedings of the USENIX Security Symposium*. USENIX Association, Baltimore, MD, 903–918.
[64] Wei Xie, Chao Zhang, Pengfei Wang, Zhenhua Wang, and Qiang Yang. 2021. ARGUS: Assessing Unpatched Vulnerable Devices on the Internet via Efficient Firmware Recognition. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. ACM, 421–431.
[65] Zynamics. 2021. BinDiff. https://www.zynamics.com/bindiff.html

A CLUSTERING AND CLASSIFICATION

Figure 6 provides auxiliary information for our classification and clustering based on AVClass2 and Vhash. As discussed in Section 6, we used BinDiff [65] to compare binary similarity on the packed and unpacked URLhaus dataset.



(a) Coloured by Vhash



(b) Coloured by AVClass2

Figure 6: Results of pairwise similarity calculation of the URLhaus samples using Ghidra 1.9.2 and BinDiff 6. 1,725 out of 2,298 samples could be analyzed, shown are only samples with a similarity score and confidence above 0.8. Colours indicate distinct clusters of Vhash and AVClass2 respectively. For Vhash, dark teal indicates a missing Vhash while for AVClass2 dark teal indicates a singleton class.