The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle

Omar Alrawi, Charles Lever, and Kevin Valakuzhy, Georgia Institute of Technology; Ryan Court and Kevin Snow, Zero Point Dynamics; Fabian Monrose, University of North Carolina at Chapel Hill; Manos Antonakakis, Georgia Institute of Technology

https://www.usenix.org/conference/usenixsecurity21/presentation/alrawi-circle

This paper is included in the Proceedings of the 30th USENIX Security Symposium.
August 11–13, 2021
978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.
Abstract
Our current defenses against IoT malware may not be adequate to remediate an IoT malware attack similar to the Mirai botnet. This work seeks to investigate this matter by systematically and empirically studying the lifecycle of IoT malware and comparing it with traditional malware that target desktop and mobile platforms. We present a large-scale measurement of more than 166K Linux-based IoT malware samples collected over a year. We compare our results with prior works by systematizing desktop and mobile malware studies into a novel framework and answering key questions about defense readiness. Based on our findings, we deduce that the required technology to defend against IoT malware is available, but we conclude that there are insufficient efforts in place to deal with a large-scale IoT malware infection breakout.

1 Introduction
The Mirai botnet set a record for the largest distributed denial of service (DDoS) attack and drew the attention of many security professionals [1]. In the aftermath of the attack, many new developments have shaped the IoT malware ecosystem. Therefore, studying the threat lifecycle for IoT malware is vital for securing IoT devices. For example, the Mirai botnet infected devices by using default usernames and passwords, but current IoT malware variants target unpatched vulnerabilities. We seek to study how the emerging IoT malware ecosystem has evolved since Mirai and whether current defenses for traditional malware can protect against it.

To investigate this matter, we need to systematically understand how IoT malware infect systems, deploy payloads, persist on systems, abuse resources, and operate their infrastructure. We guide our analysis by answering two research questions (RQ):
• RQ1: Is IoT malware different than traditional malware?
• RQ2: Are current anti-malware techniques effective against IoT malware?
To answer RQ1, we compare the IoT malware lifecycle with traditional malware and highlight the similarities and differences. For RQ2, we qualitatively evaluate how traditional anti-malware techniques work and judge their efficacy based on empirical observations from the IoT malware ecosystem.

Answering RQ1 allows the security community to understand the evolutionary trend of IoT malware and respond accordingly. For example, how do malware adapt to infect and persist on IoT devices? Are there trends that can allow us to better predict the impact of IoT malware on future IoT technologies? Compared to desktop and mobile malware, are IoT malware capabilities bound by the device’s resources? How does the IoT malware ecosystem impact different stakeholders? Furthermore, RQ2 allows the security community to gauge if there are sufficient defensive techniques to counter a fast-evolving IoT threat.

To date, there have been several efforts to investigate IoT malware [1]–[8]. However, these efforts either focus on in-depth analysis of a single malware family or rely on small malware corpora collected over short periods. Nonetheless, these efforts provide a fascinating glimpse into the IoT threat landscape and demonstrate the need for additional research. Moreover, current threat frameworks are either too complex with a focus on traditional malware, such as the MITRE ATT&CK [9] framework, or study only the infection stage of IoT malware [10], [11]. Our work seeks to address these gaps with a comprehensive evaluation of the IoT malware lifecycle. We guide our study by a principled framework that characterizes various stages of an IoT malware’s lifecycle, and we compare our findings with traditional malware.

Our work makes four contributions. First, we propose a novel analysis framework that captures the threat lifecycle of IoT malware, which considers the infection vectors, payload properties, persistence methods, capabilities, and C&C infrastructure. Second, we use our framework to systematize 25 papers that study traditional malware. Third, we characterize IoT malware by examining more than 166K samples spanning 6 different system architectures collected over a year. Fourth, we make available the largest and most comprehensive IoT malware corpus to date and include their analysis artifacts, which can be found at https://badthings.info.
Our results show that IoT malware differs from traditional malware in a few key areas including infection vectors and C&C communication. We find signature-based detection lacks support and coverage for many IoT malware variants and that at least 15% of new variants utilize packing to evade detection. Additionally, IoT malware uses various persistent methods to overcome read-only file systems found in IoT devices by reusing vendor-specific tools. We find a large array of capabilities that have been incorporated into IoT malware such as proxy services, device bricking, and information theft. We observe that the current IoT malware ecosystem has not reached its full potential but may become a severe threat due to the sheer number of IoT devices coming online. We conclude with a set of recommendations for different stakeholders including device owners, device vendors, and ISP operators.

2 Background and Related Work

Malware targeting embedded Linux-based systems was first reported in 2008 with the discovery of the Hydra IRC bot [12]. Since then, several other bots have entered the scene with various capabilities. Such bots include psyb0t [13], Chuck Norris [14], Carna [15], Tsunami [16], Aidra [17], Dofloo [18], Gafgyt [19], Elkn0t [20], XOR.DDoS [21], WiPatch [22], TheMoon [23], LUABot [24], Remaiten [25], NewAidra [26], and Moose [27]. Each family had different purposes such as credential theft [27], cryptocurrency mining [28], device destruction [29], internet-wide scanning [15], and cleaning up infected devices [2], [22]. IoT malware development has many considerations due to the heterogeneity of devices. For example, an IP camera and a set top box can have different processors, C libraries (uclibc, musel, glibc), and kernel versions/features (Linux 2.6, 3.2, 4.6, etc.).

The release of Mirai’s source code and recent developments in embedded toolchains has made it easier for IoT malware development. Antonakakis et al. [1] note that Mirai had a wide impact due to the fact that its small code base runs on diverse devices, spreads efficiently, and targets a large number of insecure IoT devices on the internet [30], [31]. The Mirai botnet took down critical DNS infrastructure [32], disconnected over 900K internet subscribers [33], and attacked a large cloud service provider [34]. Soon after the release of Mirai’s code, many variants began to surface with enhancement to its infection vector, payload obfuscation, and command-and-control (C&C) communication. For example, Satori [35], a Mirai variant, gained momentum as it exploited a new vulnerability in Huawei routers. These recent developments provide further motivation to understand the IoT malware landscape.

Prior studies looked at IoT malware from different perspectives. Cozzi et al. [36] investigate Linux-based malware but only examine 10K samples, of which 35% are for x86 and x86_64 architecture. Other studies examine specific malware families such as Mirai [1] and Hajime [2]. More comprehensive studies examine individual components of the IoT malware lifecycle. For example, several works [3], [5], [10], [37] examine IoT malware infection tactics and the payload properties. Other works [6], [38] look at how to detect IoT malware by studying its binary static structural features. De Donno et al. [11] organize IoT malware attack capabilities into a taxonomy while Choi et al. [4] study the role that C&C infrastructure plays in the lifecycle of IoT malware.

Additional efforts [39], [40] investigate scanners on the internet to identify if they are infected by IoT malware. Finally, Çetin et al. [41] present a unique perspective on IoT malware infection cleanup by combining multiple data sources and a user study to measure remediation efforts. Our work differs in two aspects, first we propose a five-component framework that captures the entire lifecycle of IoT malware, which we use to compare with desktop and mobile malware. Second, we conduct the largest and most comprehensive empirical measurement for more than 166K Linux-based IoT malware samples collected over an entire year.

3 Framework and Methodology

Next, we describe the data sources, methodology, and the framework that we use for the comparative analysis. We define each component’s subcategories and present a summary of our results in Table 1. Appendix A presents an extended analysis of desktop and mobile malware from prior works.

3.1 Comparative Framework

Our framework looks at five components for malware’s lifecycle. We study the infection vector, the payload properties, the persistence methods, the capabilities, and the C&C infrastructure. For each component, we identify techniques discussed in the literature for traditional malware (desktop/mobile) and empirically measure it for IoT malware. The following defines each component:

- **Infection Vector** is how the malware attacks a system.
- **Payload** is the dropped malware code after exploitation.
- **Persistence** is how the malware installs on a system.
- **Capabilities** are the functions in the malware code.
- **C&C Infrastructure** is how the malware communicates with the operator.

We study 25 papers from prior works to qualitatively derive subcategories under each component, which are in Appendix A. For example, we cite the work of Holz et al. [42] to support the use of drive-by downloads in desktop malware and their distribution networks. Moreover, we use the MITRE ATT&CK taxonomy to derive additional subcategories that are not found in prior work but are documented by security companies. Table 1 summarizes the comparative analysis.
3.2 Data Sources

We list all the dataset sources for our measurements in Table 2.

**VirusTotal.** VirusTotal (VT) is a malware analysis and sharing platform that is used by hundreds of commercial security companies and thousands of researchers. We source our dataset from VT and assume that it provides good coverage because of the sheer size of files submitted to the platform, see Figure 1. We use VT to identify new binary submissions that meet the following criteria: (1) ELF binaries, (2) never seen by VT before, (3) machine architecture is not x86 or x86_64, (4) ELF binary is not Android type, (5) submission is not tagged as “shared-lib,” “coredump,” or “relocatable,” (6) file size is less than 30MB, and (7) has at least one anti-virus (AV) detection. We choose this criteria based on the access limitation (10K files/day) and the following assumptions.

First, our work studies malware that target embedded IoT systems. The vast majority (82%) of IoT systems rely on Linux-based OS (ELF) [43] and utilize Reduced Instruction Set Computers (RISC) architecture [44], whereas x86 and x86_64 are based on Complex Instruction Set Computers (CISC) architecture that are mostly found in servers, desktops, and laptops. We exclude x86, x86_64, and Android malware because (1) they are well covered in prior works [45]–[49], (2) are more likely to target mobile or traditional computing devices (servers, desktops, and laptops), and (3) their volume inundate our access capacity, as shown in Figure 1.

Second, we found ELF files larger than 30MB to be mostly coredump 1, shared-lib, or relocatable 2. We found seven files, over 30MB, detected by one or more AV engine and one file detected by five or more AV engines 3. Third, our analysis pipeline can analyze native ELF binaries, therefore, it does not support Java-based Android apps, but it supports files that run on the Android Runtime environment (native). VT classifies files that run natively in Android (Android Runtime) as ELF files because Android uses a tailored version of the Linux Kernel. We found a limited number of files for Android IoT and TV, specifically, 113 (AV labels 11 as malicious) and 57 (AV labels 6 as malicious) files, respectively.

We rely on AV detection as a way to identify possible malware, similar to prior works [50]. First, we collect files with

---

1 A recorded state of a program during a crash
2 An object file that linkers use to build an executable.
3 MD5: 3c5a75bd1d81c6d55b3ed6f1729507, Label: BitCoinMiner
one AV detection to stay under the daily access quota (10K/day, see Figure 1). Second, we filter files with less than five AV detections to suppress false-positives, which are common in VT [50]. These criteria filter out possible irrelevant samples that are not likely to be IoT malware with minimal impact on the empirical results. However, we do acknowledge this might lead to a bias in the malware dataset since our collection relies on AV detections that can have inherent limitations.

**Active and Passive DNS.** Our active DNS (aDNS) dataset comes from the ActiveDNSProject [51], which actively resolves many popular zones (COM, NAME, NET, ORG, BIZ, etc.), top sites from the Alexa Top 1M, and public blocklists daily. The passive DNS (pDNS) is an anonymized dataset provided by a large internet service provider (ISP) based in the US. The ISP operates a large set of geographically-distributed local DNS resolvers that service over 40 million interconnected devices, which include IoT devices. We use aDNS and pDNS to investigate IoT malware infrastructure. Our aDNS and pDNS datasets cover the period from May 2019 up to Jan 2020. We specifically use aDNS and pDNS to enumerate relationships between observed IPs and domains. We use pDNS data to quantify the lookup volume and the number of anonymized clients resolving the C&C infrastructure.

**Bad Packets Honeypots.** Bad Packets [52] operates a set of proprietary honeypots that monitor emerging cyber threats targeting enterprise networks, IoT devices, and cloud computing environments. We provided an aggregate dataset that spans the entire month of June 2019. We use the honeypot dataset to identify attack characteristics observed on the internet and quantify what devices IoT malware target. Specifically, we use aggregate statistics about internet scans that are classified as IoT malware by Bad Packets.

**Tranco Top Site Ranking.** We use Tranco’s top site ranking [53] to identify and filter benign domains. Our static and dynamic analysis yield large sets of domains and IPs, which may not be related to malware. For example, a link to the UPX packer website is commonly found in samples that are packed by UPX.

### 3.3 Analysis Methods

Figure 2 presents an overview of our analysis and measurement methodology. We use static, dynamic, and network analysis. We do not claim any of the techniques as a novel contribution, instead, we use them as a means to study IoT malware. We rely on well-established approaches from prior works [36], [54]–[56] and tailor them for our analysis.

**Metadata Analysis.** We use VT for AV detection, AV labels, and in-the-wild names. We combine the AV labels with AVClass [57] to consolidate the labels for each sample. This metadata analysis provides context about the malware samples and helps us to correlate the findings from static and dynamic analysis.

**Static Analysis.** The goal of static analysis is to identify each binary’s target architecture, linking method (static vs dynamic), anti-analysis tactics, packing, embedded domains and IP addresses, and infection vectors. We use a set of tools from binutils suite to perform static analysis, namely readelf, objdump, objcopy, strings, and hexdump. The file tool parses the binary information and identifies the target architecture, endianness, and linking information based on the file headers. Next, we examine the ELF binaries for anti-analysis artifacts by using four heuristics. First, we inspect the ELF file for the first LOAD (PT_LOAD) segment in the section headers that is marked for read, write, and execute (RWE). This anti-analysis trick is commonly used to hide the program’s entry point and break analysis tools.

Second, we examine the ELF file for fake section headers that overlap the program’s entry point by iterating through each segment and section. For each segment, we check if the segment overlaps the entrypoint address. If we detect an overlap, we conclude that the sample has anti-analysis artifacts. This well-known tactic overlays fake data and text sections with opposite flags (switching W and X) to confuse analysis tools by parsing the fake data sections for code. Third, we examine the ELF file for fake dynamic symbol tables by checking the section header for one or more dynamic symbol tables (SHT_DYNINFO). We iterate through each
segment and look for dynamic symbol tables that come after the dynamic table (DT_SymTab) and check if the dynamic symbol table overlaps the dynamic table (virtual address + size is outside the segment). This anti-analysis technique inserts fake dynamic symbol tables for dynamically-linked binaries that mix up the symbols of functions.

Fourth, we iterate over each segment and check the section header fields (e_shoff, e_shentsize, e_shnum, e_shstrndx) for zero values. This technique removes critical information about the section headers making it impossible to parse. The Linux kernel does not use the section headers when loading and executing the ELF file, therefore removing the section headers breaks some analysis tools that rely on section headers, but does not affect the execution of the binary. Next, we try to detect UPX packed samples by looking for UPX sections and string artifacts. For UPX packed files, we also check if the UPX header is zeroed out, which usually breaks the UPX decompression utility but not the executable. We then attempt to unpack each sample using the UPX utility. Some files fail to unpack due to corrupt UPX headers, but they execute in the dynamic analyzer.

Finally, we use static analysis to extract IP addresses and domains using strings with default settings and regular expressions. For captured domains, we use tldextract, a python library, to check for properly formed domain names. For IP addresses, we remove all bogons and invalid IP addresses. We also use static analysis to identify infection vectors by using over 200 Yara signatures. We source our Yara signatures by enumerating a set of IoT and router device vendors, crawl the NVD [58], and identify Common Vulnerability and Exposure (CVE) entries that have public proof-of-concept (PoC) code. We then manually build and verify each Yara signature. For each matched Yara signature, we verify that (1) the offset matches the signature inside the binary and (2) the binary offset is referenced by the code section.

**Dynamic Analysis.** We build architecture-specific virtual machines that execute each sample and collect their system call and network traffic, which we call full-system analysis. We run each sample for 60 seconds and collect system call traces using strace and network traces. Further, we use a binary emulator that emulates the instructions and system calls of an ELF file to generate system call traces, referenced as Zelos [59] in Figure 2. The run time of a sample influences the observed behavior as documented in prior works [60]. To account for this limitation, we measure trace divergence between full-system and binary emulation. Binary emulation allows us to skip over sleep system calls and fast forward the execution of malware hence revealing possible hidden behavior. Additionally, we use leaked source code from various IoT malware found online [61] and match them with the execution traces and function symbols to identify capabilities.

We empirically found full-system emulation traces to match 85% of binary emulation traces for ARM. The remaining 15% could not be compared due to application binary interface (ABI) mismatch during full-system analysis or failure to run in binary emulation (missing required libraries or incompatible architecture version). Furthermore, we found that before 30 seconds of full-system emulation about 95% of malware will engage in network system calls that either block or loop infinitely. Hence, we chose 60 seconds to balance between analysis quality and performance. We count successfully executed samples by two metrics, namely system artifacts and network artifacts. For system artifacts we consider a malware to be active if it creates three or more processes in the VM or if it invokes 100 or more system calls.

These parameters were conservatively chosen by examining diverging traces from full-system and binary emulation. For network artifacts, we collected network traffic from the VM for 72 hours without executing any malware. We then filter out any traffic that matches the baseline or bogon networks. We note that this is a modest attempt to build a dynamic malware analysis system for six different architectures and we recognize the challenges that are documented by earlier works [54], [55], [62]. Nevertheless, we report the results in Table 3 and make our analysis tools public for the community. Dynamic analysis allows us to study infection attempts, persistence methods, exercised capabilities, and C&C communication. We use these findings to empirically document them in the lifecycle framework and compare them to desktop and mobile malware.
Infrastructure Analysis. We use a three-tiered process to filter and identify C&C indicators. First, we use Tranco [53] top sites to enumerate a list of benign domains. We count the most referenced domains and filter them using the top site list. Second, we manually inspect the new list to remove the remaining benign domains. Third, we build a bipartite graph between domains and IP addresses to find connected components and filter out additional benign clusters [56]. After removing all the benign indicators, we use historical pDNS and aDNS to expand on the malicious indicators to find common infrastructure. For IP addresses, we look into pDNS and aDNS to identify associated domains. We repeat our method on the newly identified domains and IP addresses until we remove all benign nodes. We verify each node manually.

4 Measurement Results

Using the proposed lifecycle framework, this section presents the results from our empirical measurements and observations. We summarize the results for each subsection by takeaways (TA) to help answer our research questions (RQ1 and RQ2).

Measurement Setup. We filter our dataset from 166,772 to 138,329 samples that are detected by five or more AV engines. We then analyze each sample statically and dynamically to group the results by architecture as shown in Figure 2. We use binutils, Yara, Ghidra, and hexdump to identify the target architecture, library linking, symbols, packing, and anti-analysis artifacts. For packed samples, we attempt to unpack them using UPX [63]. For dynamic analysis, we use Buildroot [64] and QEMU [65] for full-system analysis and Zelos [59] for binary emulation. We build our full-system virtual machines (VM) by using the results from static analysis to identify a common set of required libraries to include in the VMs. However, we were not able to build a VM for M68K architecture due to legacy code incompatibility, therefore, we only considered the M68K samples for static analysis.

Table 3 summarizes our analysis results by architecture. The VT metadata has two main columns, namely detection and honeypot. Detection refers to the number of samples that are detected by five or more AV engines and honeypot refers to the number of samples seen by the VT honeypot. The static analysis section has three columns, namely library linking, anti-analysis, and polymorphic. The library linking column presents the number of static and dynamic linked samples, the anti-analysis column presents the number of samples that break static analysis tools, and the polymorphic column presents the number of packed samples and how many were unpacked. Lastly, the dynamic section has two columns, namely system and network. The system column reports the number of samples that create three or more processes or invoke at least 100 system calls. For the network, we report the number of samples with DNS and outbound internet traffic.

4.1 Detection and Labeling

In Table 4, we present the top 10 AV labels grouped by system architectures. We use AV engines hosted by VT, which are reported to have better detection coverage than their desktop versions [50]. However, Figure 3 suggests that traditional AV engines lack support and detection for IoT malware. VT hosts over 70 AV engines, but only 55 support ELF files. We observe 50% of the malware is detected by less than 25 AV engines and at most by 45 AV engines as shown in Figure 3. Furthermore, AV engines appear to detect ARM malware with better coverage, over 25% of the ARM samples are detected by at least 2 AV engines. AV engines provide an AV label coverage for at least 97% of the detected malware.

We observe that the mirai label dominates in all system architectures and accounts for 76% of the PPC samples. The next most popular label is gafgyt. The ARM samples have more diverse labels in comparison with the others. For example, the label lootoor and dvmap are only found in the ARM dataset. Some labels are exclusive to a set of architectures like hajime. Herwig et al. [2] report that Hajime malware is only built for ARM, MIPS, and MIPS-EL, which is aligned with our findings. The inconsistencies in AV detection and labeling are also reported in prior studies [66], [67].

TA1. Given that no host-based intrusion detection systems (HIDS) run on IoT devices, detecting malware after an infection is not possible. However, signature-based scanners can detect suspicious binaries forensically captured from the network or the device. Our findings suggest that many AV scanners lack support or have limited signature coverage (mostly mirai labels) for IoT malware in the wild.

4.2 Infection Analysis

We observe that IoT malware use remote exploitation and default credentials to infect devices. We present a timeline in Figure 4 that shows the incorporation of exploits in IoT malware based on reports from researchers. The timeline begins right after the Mirai source code became public and extends to the end of the malware collection period (Dec. 2019). We find nine categories of devices across 70 different exploits [68]–[84]. We observe that the number of exploits increases significantly in 2019, which target new categories of
First, we observe that the exploits affect internet-facing devices and devices behind the NAT. For example, routers and firewalls are typically internet-facing while smart home devices such as hubs should be behind a NAT device. Second, we observe that most of the vulnerability types affect network services by command injection, credential leak, or default credentials. Third, the affected device architectures are mostly ARM and MIPS, nevertheless, IoT malware appears to be architecture agnostic. Finally, we observe that certain malware families, such as miner.d, xmrig, intercepter, and stealthworker target specific devices like the Synology NAS, which suggests that some IoT malware specializes in device targeting.

**TA2.** Early IoT malware (see Section 2) relied on default credentials or a specific vulnerability to compromise internet-facing IoT devices. Our findings suggest that IoT malware has evolved to rely on a suite of exploits that target many diverse device categories not seen before, which can be either internet-facing or behind a NAT device.

**TA3.** Given most IoT devices are headless, lack a graphical user interface (GUI) or peripheral devices, all observed exploits do not require user interaction. This IoT device property allows malware to efficiently infect many devices very quickly. Additionally, the architecture agnostic nature of IoT malware may potentially make them more of a threat than earlier desktop worms.
4.3 Payload Analysis

We observe that IoT malware payloads use packing, environment keying, scripting, and cross-architecture binaries. Table 3 shows that at least 15% of the malware use packing, and we were able to unpack 78% of the packed samples. The remaining samples used anti-analysis tricks that broke the standard unpacker. We observe in dynamic analysis that IoT malware payloads use environment keying before executing. For example, we see payloads profiling the device name, CPU, and memory to check for the right environment.

We found a set of payloads that rely on script interpreters like Python and Lua for functionality. However, most payloads use the system shell for system reconnaissance and persistence. For example, various binaries invoke shell commands like `uname`, `whoami`, `ls`, `crontab`, and `os-release` to collect information about the device. We observe on exploitation that multi-architecture payloads are delivered to the device to brute force the target system architecture. For example, if the malware cannot identify the device’s architecture, they test many variants of the payload for different architectures such as ARM, MIPS, PowerPC, SPARC, SH4, and M68K.

4.4 Persistence Analysis

Before presenting the results, it is important to understand how embedded devices configure their file systems. First, most embedded devices mount their rootfs (file system) as read-only (RO). This reduces wear on flash memory, eliminates system file corruption, avoids accidental overwrites, facilitates...

---

**Table 6: Top exploits found in IoT malware binaries based on static analysis.**

<table>
<thead>
<tr>
<th>Vendor</th>
<th>CVE</th>
<th>Dev. Type</th>
<th>Vuln. Type</th>
<th>Dev. Arch.</th>
<th>AV Labels</th>
<th>ARM</th>
<th>MIPS</th>
<th>PPC</th>
<th>SPARC</th>
<th>SH4</th>
<th>M68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huawei</td>
<td>CVE-2017-17215</td>
<td>Router</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>gafgyt, ircbot, mirai, tsunami</td>
<td>10,046</td>
<td>5,527</td>
<td>2,604</td>
<td>2,352</td>
<td>2,277</td>
<td>2,226</td>
</tr>
<tr>
<td>ZTE</td>
<td>-</td>
<td>Router</td>
<td>Default Cred</td>
<td>MIPS</td>
<td>gafgyt, exploitscan, mirai, tsunami</td>
<td>3,190</td>
<td>2,038</td>
<td>912</td>
<td>728</td>
<td>735</td>
<td>724</td>
</tr>
<tr>
<td>D-Link</td>
<td>CVE-2014-8361</td>
<td>Router</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>gafgyt, mirai, tsunami</td>
<td>2,378</td>
<td>1,436</td>
<td>656</td>
<td>534</td>
<td>530</td>
<td>534</td>
</tr>
<tr>
<td>GPON</td>
<td>CVE-2018-10562</td>
<td>Router</td>
<td>CMD Inject</td>
<td>Unknown</td>
<td>gafgyt, mirai, tsunami</td>
<td>2,016</td>
<td>1,245</td>
<td>339</td>
<td>448</td>
<td>443</td>
<td>435</td>
</tr>
<tr>
<td>Zyxel</td>
<td>CVE-2016-10372</td>
<td>Modem</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>gafgyt, mirai, tsunami</td>
<td>531</td>
<td>356</td>
<td>129</td>
<td>117</td>
<td>132</td>
<td>132</td>
</tr>
<tr>
<td>Juniper</td>
<td>CVE-2015-7756</td>
<td>Firewall</td>
<td>Backdoor</td>
<td>ARM</td>
<td>gafgyt, mirai</td>
<td>413</td>
<td>250</td>
<td>115</td>
<td>95</td>
<td>77</td>
<td>82</td>
</tr>
<tr>
<td>Multi-Vendor</td>
<td>-</td>
<td>DVR</td>
<td>CMD Inject</td>
<td>ARM</td>
<td>gafgyt, mirai, tsunami</td>
<td>326</td>
<td>229</td>
<td>74</td>
<td>56</td>
<td>68</td>
<td>70</td>
</tr>
<tr>
<td>D-Link</td>
<td>CVE-2013-7471</td>
<td>Router</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>gafgyt, mirai, tsunami</td>
<td>317</td>
<td>205</td>
<td>79</td>
<td>62</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>Synology</td>
<td>CVE-2017-9554</td>
<td>NAS</td>
<td>Info Leak</td>
<td>Various</td>
<td>gafgyt, intercepter, miren</td>
<td>289</td>
<td>145</td>
<td>49</td>
<td>31</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>Zyxel</td>
<td>CVE-2017-18368</td>
<td>Router</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>gafgyt, mirai</td>
<td>191</td>
<td>105</td>
<td>48</td>
<td>41</td>
<td>43</td>
<td>38</td>
</tr>
<tr>
<td>Asus</td>
<td>CVE-2018-15887</td>
<td>Modem</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>gafgyt, mirai</td>
<td>166</td>
<td>92</td>
<td>40</td>
<td>42</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>NETGEAR</td>
<td>-</td>
<td>NAS</td>
<td>CMD Inject</td>
<td>ARM</td>
<td>mirai</td>
<td>112</td>
<td>87</td>
<td>25</td>
<td>21</td>
<td>26</td>
<td>24</td>
</tr>
<tr>
<td>HooToo</td>
<td>CVE-2018-20841</td>
<td>Router</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>gafgyt, mirai, tsunami</td>
<td>112</td>
<td>60</td>
<td>28</td>
<td>17</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>WePresent</td>
<td>-</td>
<td>Router</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>mirai</td>
<td>98</td>
<td>58</td>
<td>24</td>
<td>21</td>
<td>25</td>
<td>23</td>
</tr>
<tr>
<td>LG</td>
<td>CVE-2018-17173</td>
<td>Display</td>
<td>CMD Inject</td>
<td>ARM</td>
<td>mirai</td>
<td>98</td>
<td>58</td>
<td>24</td>
<td>21</td>
<td>25</td>
<td>23</td>
</tr>
<tr>
<td>Vera</td>
<td>CVE-2013-4861</td>
<td>Hub</td>
<td>Info Leak</td>
<td>MIPS</td>
<td>mirai</td>
<td>92</td>
<td>52</td>
<td>21</td>
<td>18</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>Belkin</td>
<td>-</td>
<td>Smart Home</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>mirai</td>
<td>88</td>
<td>50</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Multi-Vendor</td>
<td>-</td>
<td>Camera</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>mirai</td>
<td>85</td>
<td>48</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Multi-Vendor</td>
<td>CVE-2017-8225</td>
<td>Camera</td>
<td>Info Leak</td>
<td>MIPS</td>
<td>mirai</td>
<td>85</td>
<td>48</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>DreamBox</td>
<td>CVE-2017-14135</td>
<td>Media</td>
<td>CMD Inject</td>
<td>PowerPC</td>
<td>mirai</td>
<td>85</td>
<td>48</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Multi-Vendor</td>
<td>CVE-2019-3929</td>
<td>Router</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>mirai</td>
<td>85</td>
<td>48</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Oracle</td>
<td>CVE-2019-2725</td>
<td>Web App</td>
<td>IMD Inject</td>
<td>x86_64</td>
<td>mirai</td>
<td>85</td>
<td>48</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Schneider-Electric</td>
<td>CVE-2018-7641</td>
<td>Industrial/Home</td>
<td>CMD Inject</td>
<td>x86</td>
<td>mirai</td>
<td>85</td>
<td>48</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Linksys</td>
<td>-</td>
<td>Router</td>
<td>Mem Corrupt</td>
<td>MIPS</td>
<td>mirai</td>
<td>83</td>
<td>50</td>
<td>20</td>
<td>19</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>EnGenius</td>
<td>-</td>
<td>Router</td>
<td>CMD Inject</td>
<td>MIPS</td>
<td>mirai</td>
<td>68</td>
<td>64</td>
<td>13</td>
<td>12</td>
<td>14</td>
<td>13</td>
</tr>
</tbody>
</table>

Figure 4: A timeline of exploits for Mirai variants based on reports from security researchers.
device update over-the-air (OTA), and eases factory reset. Still, there are processes on the device that need write-access for passwords, configurations, and keys. Embedded devices designate a non-volatile data region and a volatile temporary file system region on the flash memory. The data region is used by processes and services to store their configurations. Malware have to consider these file system constraints to persist on the device.

We observe in dynamic analysis that IoT malware attempt to persist on the device’s firmware. We must clarify that firmware refers to the IoT device’s OS, which is a customized embedded Linux instance (unified layer, see Table 1). In many IoT devices, services run as root, which means if exploited by malware then they will gain root access on the device. We observe that IoT malware use many persistent methods by installing themselves as either a service, a startup script, a system module, or a backdoor. Some samples attempt to remount the file system with read-write permissions to persist on the roots. For example, using the command `mount -o remount` malware can remount the file system with read-write permissions. In several instances, we observe malware using vendor-specific tools such as `/bin/cfgmtd` that target Ubiquiti devices to add an SSH backdoor.

Even with volatile memory regions, we observe IoT malware using tmpfs paths to persist. On system reboot, the tmpfs paths will be wiped, which will remove the IoT malware. However, to prolong the infection, we notice that IoT malware will disable the watchdog process on devices. A watchdog process on an embedded device is a privileged process that mitigates software faults by forcing a device to reboot into a clean state. If malware causes the system to become unstable, the watchdog process will reboot the device and consequently remove the malware. For example, IoT malware will disable the watchdog process by writing the "Magic Close" value ("V") to one of the following locations `/dev/FTWDT101_watchdog`, `/dev/misc/watchdog`, or `/dev/watchdog`

**TA5.** The results suggest forensic identification of infections on a device may be difficult because malware can persist in many locations. Although IoT devices mount their file system as read-only, there appears to be many methods to overcome this limitation, which can worsen infection cleanup.

### 4.5 Capability Analysis

Initial variants of IoT malware discussed in Section 2 focused on DDoS and scanning capabilities. Our analysis shows an expanded set of capabilities found in modern IoT malware. Using dynamic analysis, we observe aggressive evasion by disabling firewall processes, access control modules, ISP remote administration, unblocking restricted domains, deleting access logs, history logs, service access logs, and modifying timestamps on files. Moreover, we observe privilege escalation attempts targeting the Android Runtime environment. We also observe data theft attempts that look for Sybase database files, collect device profiles, harvest device configurations, and enumerate system files. Perhaps the most prevalent capabilities are network scanning and spreading. Table 7 is a summary of the observed scanning and exploitation attempts, which includes a subset of the vulnerabilities found in Table 6. We do not observe direct DDoS attacks, but through static analysis, we find DDoS capabilities in the malware. We identify a set of DDoS attack functions using function symbols in the analyzed samples and match them with public malware source code. Table 8 presents a list of the DDoS functions found in IoT malware.

**Table 7: Scanning methods found in IoT malware binaries based on dynamic analysis.**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Port Number</th>
<th>Attack Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telnet</td>
<td>23, 2323</td>
<td>Dictionary Attack</td>
</tr>
<tr>
<td>ADB</td>
<td>5555</td>
<td>Android Debug Bridge Shell</td>
</tr>
<tr>
<td>HTTP</td>
<td>5555, 5555, 52869, 37215, 7547, 8080, 8081, 443, 80, 81</td>
<td>Command Injection</td>
</tr>
</tbody>
</table>

**Table 8: DDoS capabilities found in IoT malware binaries based on static analysis and leaked source code.**

<table>
<thead>
<tr>
<th>DDoS Type</th>
<th>Function Symbol Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td><code>attack_tcp_syn, attack_tcp_ack, attack_tcp_stomp, attack_method_tcp, attack_tcp_vsynack, attack_tcp_fno, attack_method_tcpfrag, attack_method_tcpall, attack_method_tcpusyn, attack_method_asyn, attack_tcp_lynx, attack_method_tcpxma</code></td>
</tr>
<tr>
<td>UDP</td>
<td><code>attack_udp_generic, attack_udp_vse, attack_udp_dns, attack_udp_plain, attack_method_udpgame</code></td>
</tr>
<tr>
<td>GRE</td>
<td><code>attack_gre_ip, attack_gre_eth</code></td>
</tr>
<tr>
<td>APP</td>
<td><code>attack_app_http, attack_method_ovh, attack_method_miscdestruct, attack_app_cfnull</code></td>
</tr>
<tr>
<td>GENERIC</td>
<td><code>attack_method_std, attack_method_generic, attack_method_misckill</code></td>
</tr>
</tbody>
</table>

Additionally, we observe from dynamic analysis device destruction attempts by IoT malware. Malware will try to delete the root directory of the file system, dbus devices, zero out MMC memory, remove configured devices on general purpose IO pins, and delete the Linux device table. Furthermore, we IoT malware will abuse device resources for cryptocurrency mining and proxy services. Malware will download open-source miners such as cgminer and attempt to lock out the device owner by removing restore tools, disabling device upgrade, and hard-coding an IP address to a specific mining pool server. We also observe attempts to set up a proxy service that configures network traffic forwarding on high ports.
Infected devices can degrade or damage IoT services not only for device owners but also for network operators and device vendors. Additionally, they can facilitate criminal activities by tunneling malicious traffic through infected devices or eavesdropping on local network traffic.

### 4.6 C&C Analysis

We observe from dynamic and static analysis that IoT malware can use P2P and centralized infrastructure for C&C communication. For example, Hajime [2] uses the Kademlia overlay network, which is a P2P protocol. We also observe some malware using the Tor network either for C&C call-back or for connecting to a cryptocurrency mining pool. For centralized infrastructure, we find that IoT malware rely on hard-coded IPs rather than domains, as shown in Table 3. We only observe 7K samples with DNS lookups, which accounts for less than 7% of the network active samples. From network traces, we gather 306 unique domains and 10,895 IPs, which have a very small overlap. This reinforces that IoT malware rely mostly on hard-coded IP addresses for C&C call-back.

Lastly, we observe that some IoT malware attempt to hide their DNS IP address resolution by using DNS TXT records.

We investigate the domains and IP addresses using the pDNS dataset. Table 9 presents the top six malware families based on the infrastructure analysis described in Section 3. We rank the rows by the number of unique client IDs found in the pDNS dataset. The columns are as follows, Labels is the AV family, Clients is the number of unique client IDs, FQDN is the number of unique fully-qualified C&C domains, IP is the number of unique C&C IPs, e2LD is the number of effective second-level C&C domains, Days is the number of distinct days the C&C was queried, Samples is the number of malware, and Cluster is the number of C&C clusters per family.

We observe that the mirai samples are the most active with 874 clients, 144 e2LD, 151 unique clusters, and 2,607 associated samples. The next largest is gafgyt, which shares 63 clusters with mirai. Also, Figure 5a and Figure 5b present the malware activity as seen from pDNS. We observe that the lookup volumes are sporadic throughout the year, then for the period from November to January, there is an uptick in lookup volume especially for the tsunami family.

### 5 In-Depth Case Studies

Motivated by our empirical results in Section 4, we take a closer look at how IoT malware reuses Mirai’s code to provide more insightful answers to our research questions.

#### 5.1 Code Reuse and Evolution

**Bugs in the Source.** During our dynamic analysis, we noticed a number of IoT malware samples failed to run in the full-system emulation. Further investigation showed that the samples would crash at the beginning of execution. These samples had their function symbols stripped and only affected the MIPS-EL and ARM architecture. We tracked down the issue to a set of faulty compilers that are used in the build script of the leaked Mirai code. These compilers were specifically for ARMv6 and MIPS-EL architecture. To reproduce the bug, we compiled a test program with the faulty compilers and ran them, but they did not crash. However, when we passed the "strip" flag to the compiler, the binaries crashed on execution. This bug was found in over 8,000 ARM samples from our dataset. Moreover, we reproduced this bug on real hardware by running the test program on two physical devices, namely a Raspberry Pi 3 (ARM) and a GLiNet 300M (MIPS) router. The physical hardware exhibited the same behavior as our full-system emulation.

Investigating additional malware samples that failed in the dynamic analyzer, we found a set of traces that crash in the middle of execution. We analyzed the crash files and found that a segmentation fault is generated when the malware attempts to hide its process name. A snippet of the code is shown at the top of Figure 6 based on Mirai’s code. However, other samples did not have this bug, which used a different version of the code shown at bottom of Figure 6. The bug is caused by a fixed length buffer used to store the process name, which only supports a maximum of 20 bytes including the path of the binary. The newer code fixes this issue by using a variable-length buffer as shown in the lower portion on line three of Figure 6.

**Corrupted DNS Resolutions.** We found a large number of malformed DNS packets from our dynamic analysis, which we initially assumed to be a misconfiguration in our analyzer. We came across a set of samples that attempt to resolve a DNS resolution. However, the use of hard-coded IP addresses make IoT botnets less resilient to takedowns. IoT malware network activities can be difficult to measure on the internet using DNS since very few samples rely on DNS.

<table>
<thead>
<tr>
<th>Labels</th>
<th>Clients</th>
<th>FQDN</th>
<th>IP</th>
<th>e2LD</th>
<th>Days</th>
<th>Samples</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>mirai</td>
<td>874</td>
<td>121</td>
<td>144</td>
<td>269</td>
<td>2607</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>gafgyt</td>
<td>682</td>
<td>121</td>
<td>144</td>
<td>69</td>
<td>269</td>
<td>2727</td>
<td>73</td>
</tr>
<tr>
<td>chachaddos</td>
<td>300</td>
<td>2</td>
<td>7</td>
<td>253</td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>hajime</td>
<td>156</td>
<td>4</td>
<td>3</td>
<td>265</td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>NOLABEL</td>
<td>132</td>
<td>44</td>
<td>158</td>
<td>24</td>
<td>269</td>
<td>41</td>
<td>29</td>
</tr>
<tr>
<td>tsunami</td>
<td>112</td>
<td>41</td>
<td>48</td>
<td>18</td>
<td>268</td>
<td>263</td>
<td>34</td>
</tr>
</tbody>
</table>

Table 9: Top IoT malware clusters grouped by AV Labels.
domain but created malformed DNS packets. These samples had very similar system traces to the original Mirai code. We investigated Mirai’s code and found an initialization bug that causes DNS queries to be malformed. Specifically, the code does not initialize the buffer where the DNS query is stored, which can contain random bytes from the device’s memory as padding. We found this bug to affect all Mirai variants [61] in our study, and it appears to contribute to IoT malware reliance on IPs instead of DNS for C&C call-back.

TA9. Since DNS resolution is unreliable for samples seen in the wild, this may explain the use of hard-coded IP addresses for C&C call-back. Furthermore, given the evolutionary trends observed in other components of Mirai’s code, a fix for the DNS resolution function can make new variants more resilient to detection, blocking, and mitigation.

5.2 Payload Hosting

Having identified the DNS bug in the Mirai code, we wanted to understand how some samples used domains. We study the lifecycle of two different IoT malware C&C infrastructure, specifically, we pick iwantalldesmoke.club and str3sser.com from the top clusters identified from Section 4.6. We manually investigate these domains using DomainTools and VT.

Str3sser Domain. The str3sser.com domain was registered by Namecheap on 2018-06-29 and was inactive for almost six months. On 2018-12-27, the domain records changed to point to Cloudflare. There were two A (104.27.181.96 and 104.27.181.96) and two NS records (liz.ns.cloudflare.com and jobs.ns.cloudflare.com) created. We speculate that these records were for initial testing before going live because of the low DNS lookup volume (average 16 lookups). After 79 days, the domain’s A (35.241.225.135 and 35.205.247.152) and NS records (dns1.registrar-servers.com and dns2.registrar-servers.com) change to point to Google cloud.

Approximately 50 minutes later, based on pDNS first seen resolution, the domain is detected and reported to URLHaus. The domain remained active based on a screenshot captured nine days later but after 14 days the A records changed to a residential IP address (72.5.65.111). Finally, after two days, the owner created five child labels (cuteguys, est1976, apnewager, chivethethrottle, and ag) pointing to OpenDNS infrastructure (146.112.61.107) before the domain went offline. We base the shutdown evidence on the abrupt change in pDNS lookups from hundreds a day (average 350 lookups) to zero. The domain remained dormant with no lookups seen by pDNS sensors until it expired. The domain was used for hosting the IoT malware payload, which is downloaded after exploitation. The malware sample associated with this domain was for implementing a call-back with the C&C server using the hard-coded IP address 35.242.254.121 on port TCP/443 (not TLS). In this case, the payload domain operated for approximately 16 days.

IWanAllTheSmoke domain. The iwantalldesmoke.club domain was registered by Namecheap on 2019-01-10. A day later, one A record (185.141.24.211) is added to point to a virtual private server (VPS) (Host Sailor Ltd.). Two days later, a screenshot of the domain’s front page reads “me nah wan
go jail." On day three, 11 lookups are seen by pDNS and the domain goes dormant with no activity for five days. Then, on 2019-01-21 the domain updated the A record (89.46.223.195) to point to another VPS (Zare.com). Approximately 50 minutes later, the domain is reported to URLhaus. The domain’s DNS lookups increased to an average of 10 lookups per day, but three days later the lookups stopped. On the seventh day, the domain was no longer available and only operated for six days before going offline.

However, this domain is one of five domains associated with the payload hosting server. Using pDNS data, we observed four additional domains that were used throughout the year (Jan’19 to Jul’19) pointing to IP address 89.46.223.195 and hosting similar payloads, suggesting a rotation of payload domains. The malware sample checks-in with the C&C server using the same IP address on port TCP/9285, but instead of resolving any of the five domains the sample uses the hard-coded IP address. The domains are only used in the initial exploitation followed by payload download. These observations suggest that malware using domains for payload download rely on the device’s DNS resolution instead of Mirai’s code. Recall, many of the exploits in Section 4.2 rely on the device’s system shell to download and run the payload, hence the DNS resolution is done by the device, not the malware code.

TA10. IoT malware uses domains for payload hosting and rarely for C&C call-back. Although payload hosting domains are short-lived (i.e. six and 15 days), their lifespan is sufficient for IoT malware operation because the malware can efficiently infect many devices. This suggests that domain takedowns only affect malware spreading but not the botnet itself.

6 Summary and Discussion

Recall, RQ1 seeks to identify the similarities and differences between desktop, mobile, and IoT malware, while RQ2 seeks to qualitatively assess current defensive techniques against IoT malware.

6.1 RQ1: Similarities and Differences

First, we observe that the majority of IoT malware is based on Mirai’s code. This is vastly different from traditional desktop and mobile malware, where there are hundreds if not thousands of desktop and mobile malware families. This observation suggests that offline IoT malware detection (TA1) may be relatively easier than traditional malware because a large majority of samples in the wild stem from a shared code base. However, similar to traditional malware, polymorphism and anti-analysis (TA1) found in IoT malware can be effective in evading signature-based detection. Although we only observe 3.3% of the samples to use anti-analysis methods, we can only claim a lower bound.

The infection analysis (TA2 and TA3) suggests that IoT malware can be a bigger threat than traditional malware. For example, desktop malware has more categories of infection (drive-by, phishing, etc.), however, remote exploitation and default credentials for IoT malware apply to a larger set of architecture-agnostic internet-facing devices. Furthermore, we predict as IoT devices advance, repackaging, drive-by, phishing, and removable media will all be practical infection vectors that IoT malware may abuse. The payload analysis results (TA4) show that IoT malware has already incorporated advanced polymorphic and anti-analysis tactics, which suggests that we may see a wide adoption in the near future similar to desktop and mobile malware. One difference from traditional malware, which can be used against IoT malware, is the reliance on the device’s system shell, which can be disabled or limited (i.e. seccomp).

Persistent analysis (TA5) shows that IoT malware has to deal with file system constraints not found on desktop or mobile systems. Yet, the unification of user-space, kernel-space, and firmware removes layered protections found in traditional platforms, which can allow IoT malware to have privileged access to the device’s hardware. This suggests that although current persistent methods are limited, direct access to a device’s hardware can enable stealthier persistence tactics that may require device replacement to remediate. The capability results (TA6) present a spectrum of abuse that can range from infecting devices by scanning and exploitation to more sophisticated such as information theft and network traffic hijacking. The results in Table 6 show that some IoT malware families target specific devices, which suggests that we may see more tailored IoT device targeting based on the malware’s capabilities (rise of specialization). This is analogous to desktop malware that specializes in financial crime, ransomware, and credential theft, for example.

Furthermore, IoT malware C&C communication results (TA7) show a mix of P2P and centralized control infrastructure. Based on the abrupt IoT botnet activity observed on ISP networks, botnet operators may shift to implement a similar layered C&C communication approach to the Storm botnet [42] to achieve scalability, stability, and resilience. However, IoT malware reliance on Mirai’s code may have hindered its potential due to inherited bugs (TA8 and TA9). This is further evident by the fact that IoT malware operators use DNS mostly for payload hosting (TA10). It appears based on the infrastructure analysis in Section 5, IoT malware operators have adapted to register multiple domains for payload hosting. Since IoT malware uses a very noisy internet-wide scanning and infection approach, the payload domains are quickly detected and blocked. On the other hand, it seems that short-lived payload domains provide sufficient time for the botnet to spread (TA10).
6.2 RQ2: Stakeholders and Defenses

We identify three primary stakeholders, namely device owners, device vendors, and ISP operators.

**Device Owners.** Device owners have limited options for detecting and removing IoT malware infections. Device owners, whenever possible, should disable internet-facing services, change default credentials, and segment their network to mitigate some of the risk of infection. Most device owners would reboot their device if it becomes unresponsive or the quality of service degrades, which is also applicable to IoT malware infections. Although most IoT malware may be cleaned up with a simple reboot, we have observed several instances of IoT malware using more persistent methods (TA5). Moreover, re-imaging the device with a trusted firmware may not be possible, is technically difficult, or can damage the device. We believe the impact of this problem is much more serious than reported in prior work [41]. Specifically, we speculate that the current reinfection rates are much higher than what was measured in 2017/2018 (only 5%).

**Device Vendors.** Device vendors have end-to-end visibility that can provide early detection and remediation of IoT malware infections. For example, device telemetry can help detect system anomalies, device firmware can limit system shell interaction, containerization can limit cross-process interaction, process whitelisting can allow only trusted processes to run, remote attestation via trusted execution can guarantee a clean state, and client-server design can limit the exposed services on the network, therefore reducing the attack surface. These approaches may not all be cost-effective for vendors, but some features can be implemented as default protections for embedded Linux to boost the overall security of Linux-based IoT devices. Moreover, as vendors innovate in the IoT space, they must be mindful of future attack surfaces. For example, future IoT devices may incorporate more human interactions, which can inherit all the attacks from traditional malware such as phishing, drive-by download, and application repackaging.

More precisely, incorporating a browser in an IoT device allows IoT malware to reuse attack tactics that are found in traditional malware.

**ISP Operators.** ISP operators can play an important role in IoT malware infection cleanup as documented by Çetin et al. [41]. Besides using a walled garden for infected customers, ISPs can hinder the infection by deploying IP blocking and redirection for known IoT C&C or payload hosting servers. A more active approach would be for ISPs to intercept payload delivery or C&C communication and instead deliver a therapeutic payload that cleans up and disables vulnerable services transparently without the user involvement. However, this approach requires careful planning and engineering to scale to large networks. Current defenses at the ISP level can disrupt IoT malware infection breakouts, but this requires close monitoring and measurements to detect such events.

7 Conclusion

This work provides a large-scale empirical measurement of the current IoT malware threat landscape. By analyzing over 166K Linux-based IoT malware, we uncovered important insights that compare and contrast traditional desktop and mobile malware to IoT malware. We find that IoT malware evolution follows a similar lifecycle trend to traditional malware by using exploits for infection, packing its payload to avoid detection, using specialized capabilities based on device resources, and leveraging P2P and centralized infrastructure for C&C call-back. We speculate that IoT malware will be a much more serious threat because of the number of new IoT devices that come online and the unrealized potential of IoT malware development. Based on our findings, we believe that the required technology to defend against IoT malware is available. However, we do not think there are sufficient preparation efforts to proactively deal with a large-scale breakout. In effort to support this ongoing research in the IoT malware space, we release the largest IoT malware corpora to date and make our tools, analysis artifacts, and results available at: https://badthings.info.

8 Acknowledgment

We thank the anonymous reviewers, Ranjita Pai Kasturi, Dr. David Mohaisen, Dr. Roberto Perdisci, and Dr. Brendan Saltaformaggio for their help in improving this work. We thank Bad Packets LLC for sharing their data. This work is supported in part by the US Department of Defense grant no. FA8750-17-C-0016. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. Department of Defense (DoD).

References


B. P. LLC, Bad Packets - We provide cyber threat intelligence on emerging threats, IoT botnets, and network abuse, [Online; accessed 25 January 2020].


National Institute of Standards and Technology (NIST), NATIONAL VULNERABILITY DATABASE, [Online; accessed 25 January 2020].

Zeropoint Dynamics, Zeol: A comprehensive binary emulation and instrumentation platform, [Online; accessed 30 September 2020].


This section provides an extended analysis of malware threats for desktop and mobile platforms to compare with IoT malware in Table 10.

### A.1 Infection Comparison

#### Desktop Infection Vectors.

In Table 10, we see desktop malware pioneered many of the infection techniques. Moore et al. [85] document the SQL Slammer worm that exploited vulnerable SQL services on the internet. Although no large academic study explored desktop malware use of repackaging, default credentials, and removable media, there are ample instances from security companies documenting these techniques [104]–[106]. Desktop malware rely more on infection vectors like drive-by-download and phishing. Provos et al [93] present an extensive study on drive-by-downloads, and several prior works measure [42], [93], [97], [100] and propose defenses [94], [95], [99] for them.

For phishing, Abu Rajab et al. [87] present a multi-dimensional measurement into botnets. Their work documents how botnets leverage phishing emails for spreading. Holz et al. [42] and Kotzias et al. [46] empirically show that phishing is a common infection vector affecting desktop users. Desktop malware continued to evolve and make up a large portion of the threats on the internet. The key insight is that desktop malware initially used remote exploitation and default credentials to automatically spread but has evolved to depend on user interaction. Currently, desktop malware’s most common infection techniques require user interaction such as phishing (email), drive-by-download (browsing), removable media (physical interaction), and repackaging (i.e. pirated software).
Table 10: A comparison between desktop, mobile, and IoT malware using the proposed framework.

<table>
<thead>
<tr>
<th>Components</th>
<th>Summary</th>
<th>Desktop</th>
<th>Mobile</th>
<th>IoT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categories</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote Exploit</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Repackaging</td>
<td>✓** ✓+</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Drive-by</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Phishing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Default Cred</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Rem. Media</td>
<td>✓** ✓+</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Firmware</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>OS - Kernel</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>OS - User</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Priv. Escalation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Info. Theft</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Scanning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>DDoS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Destruction</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Resource Abuse</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>C&amp;C</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Peer-2-Peer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Centralized</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Email/SMTP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

* Techniques documented by security researchers. + Unified software layer that integrates OS and firmware.

Mobile Infection Vectors. Similar to our study, Zhou et al. [47] look at Android mobile malware and characterize the infection techniques. Their work shows that many Android malware use repackaging, drive-by download, and phishing to propagate as shown in Table 10. Lindorfer et al. [48] identify removable media propagation techniques in their large-scale study. The key insight is that unlike desktop malware, mobile malware is dependent on user interaction. Automated spreading has not been documented for the mobile platform. While worm-based malware for the Android platform do exist, they require users to visit a link to get infected.

A.2 Payload Comparison

Desktop Payload Properties. In Table 10, we see that all the payload categories apply to desktop malware. Kruegel et al. [86] predicted the rise of polymorphic payloads and proposed a way to detect them offline. Later, Barford et al. [88] studied the operation of several desktop family bots, such as GT bot, SpyBot, SDBot, and Agobot, and identified polymorphic payload obfuscation using XOR encoding. Moreover, Holz et al. [42] show that the payloads for the Storm botnet are polymorphic and change every minute, which ensures the payload has different static features to evade detection. Rossow et al. [98] studied downloaders, which are bots that download other malware or unwanted programs. Their work identified more than eight different packer techniques in use by downloaders. These findings suggest that desktop malware payloads use polymorphism to evade detection.

On the defense side, Invernizzi et al. [99] propose a technique to detect polymorphic payloads in large networks by augmenting networking information such as URI and counts. In addition to packing, environmental keying [60], [107] and scripting [108] are key components for desktop malware to bypass network and host defenses. For scripting, the payload is in the form of a text file that is executed by an interpreter such as Powershell, Python, Lua, or sh. Moreover, desktop malware makes use of cross-architecture and platform payloads for banking malware [48]. These observations suggest that the packaging of cross-architecture and platform payloads introduce a novel infection approach by crossing from trusted devices such as mobile phones and desktops.

Mobile Payload Properties. Zhou et al. [47] observe polymorphic and environmental keying behavior in Android apps. They identify malware samples that adopt the use of polymorphic techniques in the Android environment by using code reflection. They also identify malware samples that check the integrity of their code to ensure that the code is not tampered with. Similar to desktop malware, Lindorfer et al. [48] observe Android malware embedding Windows malware with autorun features that execute once the phone is plugged into a desktop. This advanced behavior leads to cross-architecture and platform infection from trusted devices giving attackers further reach. The key insight is that mobile malware use the same techniques as desktop malware but have limited script-based payloads. Script payloads for mobile devices can be invoked from installed applications, WebView, or exposed services like Android Debug Bridge (ADB), which requires

USENIX Association

30th USENIX Security Symposium 3521
the malware to be already present on the device.

A.3 Persistence Comparison

Desktop Malware Persistence. Table 10 shows that desktop malware use all levels of persistence. Provos et al. [93] and Polychronakis et al. [91] identify bots that persist through user-space and kernel modules, respectively. Additionally, Stone-Gross et al. [94] document torpig’s botnet and the mebroot infector, which both modify the Master Boot Record (MBR) entry on a hard drive’s partition allowing them to run before the OS. Desktop malware demonstrate the capability to persist on machines at many levels from the user-space all the way down to the firmware, which are outside the visibility of security tools making them hard to detect and remove.

Mobile Malware Persistence. Mobile malware by default installs and persists as a mobile app on devices unless removed by users or security software. Mobile malware can request background service permissions, subscribe to activities, and broadcast receivers giving it multiple entry points for execution. Researchers [47], [48], [103] show that mobile malware leverage all these entry points for persistence on the Android platform. For example, if malware subscribes to a broadcast receiver for SMS, the malware can execute a specific code that reads the SMS content. The key insight is that the event-driven nature of mobile applications provide a unique persistence method for malware. Detecting event-driven methods is more challenging because it requires anti-malware tools to know the triggering event ahead of time, which can be difficult when the malware is obfuscated.

A.4 Capability Comparison

Desktop Malware Capability. In Table 10 we find that desktop malware exhibit all of the listed capabilities. Moore et al. [85] document the capabilities in the Slammer worm, which other botnets also borrow [46], [87], [88], [91], [97]. Several works [92], [94], [96], [101] identify information theft and resource abuse (cryptocurrency mining, click fraud, proxy services, spam, etc.) as a common use of infected devices by desktop malware. Additionally, more recent activities include ransoming devices [46] and DDoS attacks [87] for hire.

Another aspect of desktop malware capabilities is the fact that it can escalate privileges [91] by exploitation or key-logging, and they can evade detection by disabling security tools [60], [98]. The key insight is that desktop malware have diverse capabilities, and malware families specialize based on the intended target and the attacker’s goal. For example, remote access can be a specialized capability that targets payroll processing systems. Moreover, the amount of sensitive information and compute resources (i.e. GPU) found on desktop platforms may make them a desirable target for ransom, information theft, extortion, and compute intensive abuse.

Mobile Malware Capability. Table 10 shows that mobile malware has the same abusive capabilities as desktop malware with the exception of scanning and DDoS attacks. Zhou et al. [47] identify malware that root mobile devices, evade detection through dynamic code reflection, steal sensitive information, and abuse SMS services by sending messages to premium numbers. Lindorfer et al. [48] present similar findings, but in addition they find ransomware capabilities that lock devices in exchange for payment. Mobile malware implement a subset of the capabilities found in desktop malware, which may be correlated with the features found on each platform. Unlike desktops, mobile devices generally have lower bandwidth, lower compute resources, are energy conservative, and support a single-user profile.

A.5 Command & Control Comparison

Desktop Malware C&C. Table 10 shows that desktop malware use all of the listed methods for C&C communication. Polychronakis et al. [91] show that desktop malware rely on email for C&C call-back. Moreover, Kanich et al. [92] and Holz et al. [42] study the Storm botnet P2P network to analyze the spam campaigns and estimate the botnet size. They identify a complex layered infrastructure of hierarchy of workers, proxies, and master nodes based on the Kademlia DHT protocol. They speculate that this complex infrastructure allows the botnet to scale and be resilient to takedowns. However, Rossow et al. [98] found from a large-scale study that centralized infrastructure was more prevalent than P2P.

For centralized C&C infrastructure to be more resilient, malware use domain generation algorithms (DGA) [94], [99], multi-tier centralized topology [96], fast-flux [45], and bullet-proof or hacked [101] servers. The key insight is that desktop malware enhances the scalability and resilience of their infrastructure by organizing into specific topologies or by incorporating pseudo-randomness in their domains. For example, Holz et al. [45] note content delivery networks (CDNs) and round-robin DNS (RRDNS) provide resilience for internet applications, which malware mimics by using fast-flux.

Mobile Malware C&C. Lindorfer et al. [48] found that even though the majority of malware use centralized C&C servers, some mobile malware use SMTP to send sensitive information by email. Most empirical measurements [47], [48], [102] identify that mobile malware does not use the same sophistication for C&C call-back found in desktop malware. Furthermore, Lever et al. [102] compared mobile malware domains with desktop malware domains and found no major differences. The key insight is that mobile malware may not use sophisticated C&C infrastructure because of their network mobility property. For example, if a mobile device is connected to a network that blocks its C&C server (mobile network operator), the device will eventually connect to another network (coffee shop WiFi) as it changes its physical location, which may allow connections to the C&C server.