Understanding Fileless Attacks on Linux-based IoT Devices with HoneyCloud

Fan Dang[1], Zhenhua Li[1*], Yunhao Liu[1,2], Ennan Zhai[3]
Qi Alfred Chen[4], Tianyin Xu[5], Yan Chen[6], Jingyu Yang[7]
[1]Tsinghua University [2]Michigan State University [3]Alibaba Group [4]University of California, Irvine
[5]University of Illinois Urbana-Champaign [6]Northwestern University [7]Tencent Anti-Virus Lab

ABSTRACT

With the wide adoption, Linux-based IoT devices have emerged as one primary target of today's cyber attacks. Traditional malware-based attacks can quickly spread across these devices, but they are well-understood threats with effective defense techniques such as malware fingerprinting and community-based fingerprint sharing. Recently, *fileless attacks*—attacks that do not rely on malware files—have been increasing on Linux-based IoT devices, and posing significant threats to the security and privacy of IoT systems. Little has been known in terms of their characteristics and attack vectors, which hinders research and development efforts to defend against them. In this paper, we present our endeavor in understanding fileless attacks on Linux-based IoT devices in the wild. Over a span of twelve months, we deploy 4 hardware IoT honeypots and 108 specially designed software IoT honeypots, and successfully attract a wide variety of real-world IoT attacks. We present our measurement study on these attacks, with a focus on fileless attacks, including the prevalence, exploits, environments, and impacts. Our study further leads to multi-fold insights towards actionable defense strategies that can be adopted by IoT vendors and end users.

CCS CONCEPTS

• **Security and privacy → Hardware attacks and countermeasures**; **Mobile and wireless security**.

ACM Reference Format:
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 *The 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '19), June 17–21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307334.3326083

1 INTRODUCTION

Internet of Things (IoT) has quickly gained popularity across a wide range of areas like industrial sensing and control [36], home automation [18], *etc.*. In particular, the majority of today's IoT devices have employed Linux (*e.g.,* OpenWrt and Raspbian) for its prevalence and programmability, and such a trend has been growing continuously [1]; meanwhile, the number of cyber attacks against Linux-based IoT devices is also increasing rapidly [5]. In this paper, we, therefore, focus on Linux-based IoT devices and the attacks targeting them. The Linux-based IoT attacks generally fall into two categories: *malware-based attacks* and *fileless attacks*.

Threats from malware-based attacks (*e.g.,* Mirai, PNScan, and Mayday) have been widely known in IoT networks. For example, global websites like GitHub and Twitter became inaccessible for hours in Oct. 2016, since their DNS provider, Dyn, was under DDoS attack by Mirai, which infected over 1.2 million IoT devices [11]. These incidents raised high awareness of malware-based attacks on IoT systems; throughout the past few years, their characteristics have been extensively studied and effective defense solutions have been developed. For instance, the hash (*e.g.,* MD5 or SHA-n) of a malware's binary file can be computed to fingerprint these IoT malware, and such fingerprints are then shared with the community such as through VirusTotal[1]. For a malware that has not been fingerprinted, static and dynamic analysis can be applied to determine their malice [16].

Fileless attacks (also known as non-malware attacks or zero-footprint attacks) on IoT devices differ from malware-based attacks in that they do not need to download and execute malware files to infect the victim IoT devices; instead, they take advantage of existing vulnerabilities on the victim devices. In the past few years, increasingly more fileless attacks have been reported [4, 37], *e.g.,* McAfee Labs reports that fileless attacks surged by 432% over 2017 [3]. Traditional servers and PCs defend against fileless attacks using sophisticated firewalls and antivirus tools [6]; however, these solutions are not suitable for the vast majority of IoT devices due to the limited computing and storage resources. As a result, fileless attacks pose significant threats to the IoT ecosystem, given that IoT devices are often deployed in private and sensitive environments, such as private residences and healthcare centers. At the moment, there is limited visibility into their characteristics and attack vectors, which hinders research and development efforts to innovate new defense to combat fileless attacks.

1.1 Study Methodology (§2)

To understand Linux-based IoT attacks in the wild, we use *honeypots* [31], which are known to be an effective method for capturing unknown network attacks. We, therefore, first set up several common Linux-based IoT devices in different places as hardware honeypots. Each honeypot is coupled with a Remote Control Power

[*] Corresponding author.

[1] https://www.virustotal.com

MobiSys '19, June 17–21, 2019, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6661-8/19/06…$15.00
https://doi.org/10.1145/3307334.3326083

Adapter which can reset the honeypot when it is compromised. These offer us valuable insights into the specialties of IoT attacks. However, we notice that this first endeavor incurs unaffordable infrastructure and maintenance costs when deployed at scale. We, therefore, attempt to explore a *cheap* and *scalable* approach to effectively capture and analyze real-world IoT attacks.

Intuitively, such an attempt can be empowered by public clouds widely spread across the world. This seems to be a possible host for our quickly deploying numerous software (virtual) honeypots. Nevertheless, this approach is subject to several practical issues. *First,* the virtual honeypots should behave similarly to actual IoT devices, so as not to miss the relatively rare fileless attacks. *Second,* they should expose in-depth information of the interaction processes, to facilitate our characterizing the usually hard-to-track fileless attacks. *Finally,* they have to conform with diverse policies imposed by different cloud providers, so that one cloud's limitations would not essentially influence the coverage of our study results. To this end, we heavily customize the design of software honeypots by leveraging the insights collected from hardware honeypots, such as kernel information masking, encrypted command disclosure, and data-flow symmetry monitoring. We carefully select eight public clouds to disperse 108 abovementioned software honeypots, and the system is called HoneyCloud. These software honeypots employ OpenWrt, which is one of the most popular Linux distributions for IoT devices [1] and also suitable for customization.

Compared to a hardware honeypot as we show in §2.2, a software honeypot attracts 37% fewer suspicious connections and 39% fewer attacks on average. On the other hand, the average monthly maintenance fee of a software honeypot (~6 US dollars) is 12.5× less than that of a hardware honeypot (~75 US dollars). More importantly, all the types of attacks captured by our hardware honeypots have also been captured by HoneyCloud (but not vice versa). This shows the effectiveness of our design in practice.

1.2 Findings and Implications (§3)

During one year's measurement (06/15/2017–06/14/2018), we observed 264 million suspicious connections to our honeypots, of which 28 million successfully logged in and enabled attacks.[2] Among these successful attacks, 1.5 million are identified as fileless attacks,[3] by investigating which we acquire the following key insights:

- **We introduce the first taxonomy for fileless IoT attacks by correlating multi-source information.** For lack of malware files and fingerprints, it is not an easy task to identify and classify fileless attacks: by comprehensively correlating the disclosed shell commands, monitored filesystem change, recorded data-flow traffic, and third parties' online reports, we identify the numerous fileless attacks and empirically classify them into eight different types in terms of behaviors and intents (§3.3). To our knowledge, this is the first taxonomy for fileless attacks in the IoT area.
- **Fileless attacks aggravate the threats to IoT devices by introducing stealthy reconnaissance methods and unique types of IoT attacks**. On one side, we notice that 39.4% of the captured fileless attacks are collecting system information or performing de-immunization operations (*e.g.,* shut down the firewall and kill the watchdog) in order to allow more targeted and efficient follow-up attacks. We suspect this is because fileless attacks are harder to fingerprint, and thus are highly suitable for stealthy attack reconnaissance or preparations. On the other side, we find that fileless attacks can also be powerful attack vectors on their own while maintaining high stealthiness. Specifically, we capture a fileless attack in the wild that launched a targeted DDoS attack. The attack neither modifies the filesystem nor executes any shell commands, but can manipulate a swarm of IoT devices and make the attacker(s) invisible to victims. Since the only indication of such an attack is anomalous patterns of outbound network traffic, it is highly challenging for existing host-based defense mechanisms to detect it effectively.
- **IoT attacks in the wild are using various types of information to determine device authenticity**. According to our measurements on hardware honeypots, 9132 attacks executed commands like `lscpu` to acquire sensitive system information. In addition, with HoneyCloud we find an average of 6.7% fewer attacks captured by a honeypot hosted on AWS than one hosted on other public clouds, probably because AWS has disclosed all its VM instances' IP ranges and some malware like Mirai does not infect (in fact intentionally bypasses) specific IP ranges [11]. These insights are then leveraged to improve the design and deployment of HoneyCloud in fidelity and effectiveness.
- **We discover new security challenges posed by fileless attacks and propose new defense directions.** While leaving zero footprint on the filesystem, almost all the captured fileless attacks are using shell commands and thus are detectable by auditing the shell command history of IoT devices. Unfortunately, we notice that many IoT devices use a read-only filesystem to mitigate malware-based attacks, but unexpectedly increases the difficulty (in persisting the shell command history) of detecting fileless attacks. This is a fundamental trade-off between the auditability of fileless attacks and the security against malware-based attacks, which presents a new IoT security challenge posed by fileless attacks. Moreover, we observe that the majority (65.7%) of fileless attacks are launched through a small set of commands: `rm`, `kill`, `ps`, and `passwd`, which are enabled by default in our honeypots (and almost all real-world Linux-based IoT devices). For these commands, in fact not all of them are necessary for a special-purpose IoT device, which thus creates opportunities to effectively reduce the attack surface by disabling them.

The insights above provide us with actionable defense strategies against fileless attacks, and we integrate and embody them into a practical workflow we call IoTCheck. For a Linux-based IoT device, the IoTCheck workflow can guide the manufacturers and administrators how to check its security and what to check, along with giving the corresponding defense suggestions that are easy to follow. We also release our data collected in the study at https://honeycloud.github.io, as well as the customization code for building the honeypots.

[2] The rest of connections (*i.e.,* 89.4% of the observed suspicious connections) cannot intrude into our honeypots, since they failed to crack the passwords of our honeypots.

[3] Different from previous studies on IoT attacks where fileless attacks were scarcely reported, our honeypots captured substantially more fileless attacks with diverse features. Also, we find that a root cause lies in the weak authentication issue of today's IoT devices, which makes it unprecedentedly easy for attackers to obtain remote control and then perform malicious actions without using malware.

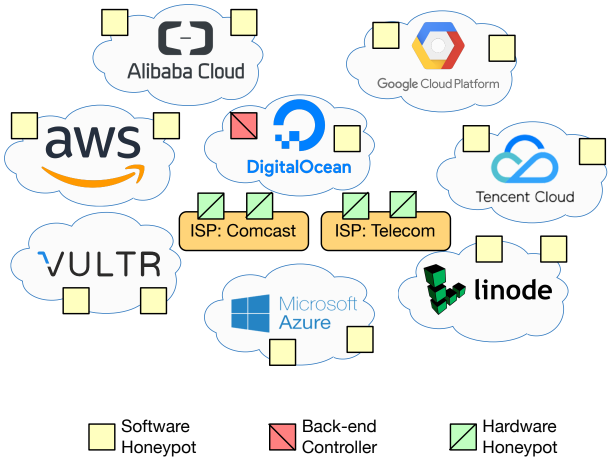


Figure 1: Deployment overview of our system.

2 HONEYPOT DEPLOYMENT

As an effective tool for understanding known and unknown attacks, honeypots have been widely used and deployed on the Internet [31]. Conceptually, a honeypot is a system connected to a network and monitored, when we *expect* it to be broken into and even compromised [28]. Given the fact that a honeypot is not a production system that aims to provide services, nobody has really decent reasons to access it. Thus, we believe that communication packets going to and coming from a honeypot should be typically *suspicious*[4].

2.1 Overview

Figure 1 gives an overview of our IoT honeypot deployments, termed HoneyCloud. It consists of both hardware IoT honeypots (§2.2) and cloud-based software IoT honeypots (§2.3). The hardware IoT honeypots are deployed at four different geographical locations as shown in Table 1. The software IoT honeypots are deployed on 108 VM instances (whose geo-distribution is depicted in Figure 2) from eight public clouds across the globe, including AWS, Microsoft Azure, Google Cloud, DigitalOcean, Linode, Vultr, Alibaba Cloud, and Tencent Cloud.

Hardware IoT honeypots. Our first endeavor is to build and deploy hardware IoT honeypots. Section 2.2 describes our design, implementation, and operation experiences. We deployed four hardware honeypots on Raspberry Pi, Netgear R6100, BeagleBone, and Linksys WRT54GS placed in residences of the team members from different cities. Table 1 lists the location, hardware configurations, and the monetary cost of the deployment.

From Jun. 2017 to Jun. 2018, the hardware IoT honeypots attracted 14.5 million suspicious connections. 1.6 million among the suspicious connections successfully logged in and were taken as effective attacks. Among these attacks, 0.75 million are malware-based attacks, and 0.08 million are fileless attacks. Note that the remaining 0.79 million cannot be classified into malware-based or fileless attacks because they did nothing after logging in.

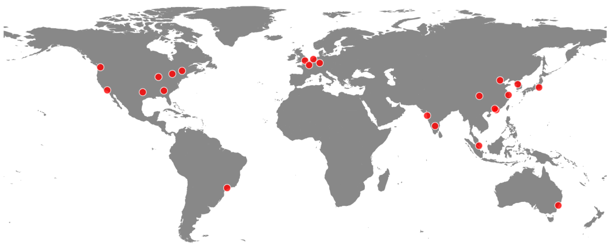


Figure 2: Geo-distribution of our deployed software honeypots.

Despite their effectiveness, hardware IoT honeypots are expensive and require high maintenance overhead. The deployment of four hardware honeypots costs 280 US dollars for devices and 280 US dollars *per month* for Internet connections. Given that the lifespan of an IoT device is typically a couple of years, the monthly infrastructure fee is around 300 US dollars. Moreover, although the deployment and maintenance schemes of the other three honeypots are almost identical to that of the first honeypot, the (manual) labor of maintenance to check low-level infrastructure dependencies cannot be avoided for the other three honeypots.

Software IoT honeypots. The insights and experiences collected by running hardware IoT honeypots drive us to build software-based IoT honeypots that can be deployed in the cloud at scale. Since Jun. 2017, we have deployed 108 software IoT honeypots at eight public clouds across the globe. As of Jun. 2018, we had observed 249 million suspicious connections to our software honeypots, of which 10.6% successfully logged in and became effective attacks. Among these attacks, 14.6 million are malware-based attacks, and 1.4 million are fileless attacks. The remaining 10.4 million cannot be classified into malware-based or fileless attacks because nothing happened after successfully logging in.

Since an IoT device typically possesses quite low-end hardware, we only need to rent entry-level VM instances to accommodate software honeypots (*i.e.,* one VM instance hosts one software honeypot). The typical configuration of a VM instance comprises a single-core CPU @ ~2.2 GHz, 512 MiB of memory, 10–40 GB of storage, and 100 Mbps of network bandwidth.

2.2 Hardware IoT Honeypots

Design and implementation. Figure 3 shows our hardware honeypot implementation, which consists of three parts: a) basic hardware including Network Interface Card (NIC), RAM, and CPU/GPU; b) the OpenWrt operating system; and c) the honeypot services for attracting attackers.

Our hardware honeypot records all actions of the attackers (including the commands typed or programs executed by the attackers), and reports these actions to the Data Collector (see Figure 3). We expose *ash*—a shell provided by BusyBox[5]—to attackers, so as to record their operations when they execute shell commands. The password of the `root` user is set to `root` by modifying the `shadow`

[4] For honeypots deployed in public clouds, the situation can be slightly different since VM instances can report diagnostic data (which are of course legitimate) to cloud providers. Fortunately, such traffic can be easily recognized and we do not consider it in our analysis.

[5] https://busybox.net

Table 1: Specifications of our hardware IoT honeypot deployment.

#	City	Device and the Price	CPU Architecture	Memory	Internet Access (per month)
1	New York, US	Raspberry Pi, US$20	ARM @700 MHz	256 MiB	US$80, ISP: Comcast
2	San Jose, US	Netgear R6100, US$55	MIPS Big-Endian @560 MHz	128 MiB	US$80, ISP: Comcast
3	Beijing, CN	BeagleBone, US$45	ARM @720 MHz	256 MiB	US$20, ISP: Unicom
4	Shenzhen, CN	Linksys WRT54GS, US$40	MIPS Little-Endian @200 MHz	32 MiB	US$20, ISP: Telecom
*	All above	Remote Control Power Adapter (RCPA), US$30	N/A	N/A	US$30 (US), ISP: Comcast US$10 (CN), ISP: Unicom/Telecom

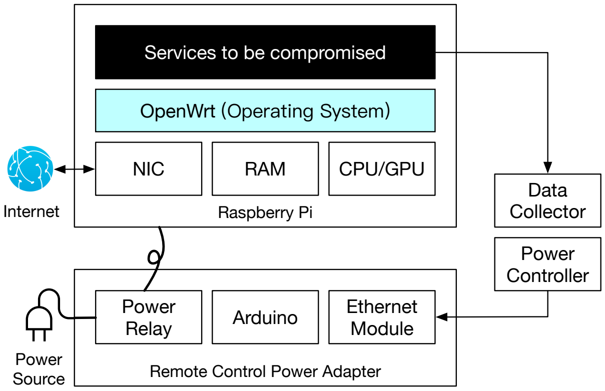


Figure 3: System architecture of our developed hardware IoT honeypot based on Raspberry Pi.

file. The SSH service is provided by Dropbear[6] on port 22 and the Telnet service is provided by BusyBox on port 23.

When we identify an attack that intrudes a hardware honeypot, we capture threat information of this attack and then *reset* the honeypot. Our implementation utilizes *initramfs*[7] to achieve an in-memory filesystem, where the filesystem is first loaded from the flash memory or the SD card and then unpacked into a RAM disk, and all modifications to the system state will be lost after the reset.

If our hardware honeypot is unavailable (*i.e.,* it does not report data to the Data Collector, or we cannot log in to it), we *reboot* the honeypot. While Linux has a built-in watchdog for automatically rebooting, we observed that attackers typically use malware (*e.g.,* Mirai) to disable the watchdog. Therefore, we build a Remote Control Power Adapter (RCPA), as shown in Figure 3, to physically reboot the honeypot.

The RCPA is made up of an Arduino, a power relay, and an Ethernet module. The RCPA uses the Message Queuing Telemetry Transport (MQTT) protocol [21] to subscribe the reboot topic from the Power Controller. Once the RCPA receives a reboot command, it triggers the power relay to power off and on the honeypot.

Experiences. Deploying and maintaining hardware IoT honeypots are non-trivial. As shown in Table 1, while building a hardware IoT honeypot only costs 20 and 30 US dollars for purchasing a Raspberry Pi and the RCPA (which are quite cheap), the Internet access fee for the two devices reaches 80 and 30 US dollars *per month* respectively (which are relatively expensive). Note that the two devices cannot share an Internet connection (*e.g.,* through NAT), because the honeypot has to be directly exposed to the Internet without NAT (otherwise, many attacks cannot reach the honeypot).

Moreover, maintaining hardware IoT honeypots incurs high overhead. In particular, the attempts to reset hardware honeypots are not guaranteed to be successful. Oftentimes, we have to manually check low-level infrastructure dependencies, such as the underlying Internet connections, the power supply, and the hardware devices. Glitches of any involved entity would increase the maintenance overhead.

Implications to software IoT honeypots. Our experiences show that hardware IoT honeypots are hard to scale due to the excessive deployment cost and maintenance overhead. To deploy IoT honeypots at scale would require software-based solutions. Our experiences reveal that the main obstacles of hardware deployments lie in the examination of low-level infrastructure dependencies. Therefore, if we can guarantee the reliability and scalability of low-level infrastructures, the issues would be effectively avoided or alleviated. This drives our efforts to build cloud-based software IoT honeypots, as cloud platforms offer probably the most reliable, scalable, and cost-efficient solution to a global-scale deployment of virtual computing infrastructure.

Meanwhile, we require the cloud-based software honeypots to possess a similar level of fidelity to the hardware IoT honeypots. Our collected measurement data from the hardware honeypots offers us useful implications for ensuring the fidelity of software honeypots. First, we find that 9,132 attacks attempted to acquire sensitive system information via commands like `lscpu` and `cat/proc/cpuinfo`, which enables attackers to detect software honeypots running on VM instances; thus, our software honeypots should be able to forge real system information (*e.g.,* CPU information), making the software honeypots look like real IoT devices (detailed in §2.3.1). Second, we notice that 187 attacks used commands like `lsusb` (listing connected USB devices) to detect potential honeypots; thus, we should enable common buses to ensure the fidelity (detailed in §2.3.1).

2.3 Software IoT Honeypots

Figures 4 illustrates our software-based IoT honeypot design. We extend the Data Collector and the Power Controller described in §2.2 to integrate our hardware IoT honeypots. This includes implementing the reset capability for software IoT honeypots in the Power Controller.

The internal structure of a software IoT honeypot consists of three modules: High Fidelity Maintainer (§2.3.1), Shell Interceptor and Inference Terminal (§2.3.2), and Access Controller (§2.3.3). The

[6] https://matt.ucc.asn.au/dropbear/dropbear.html

[7] http://www.linuxfromscratch.org/blfs/view/svn/postlfs/initramfs.html

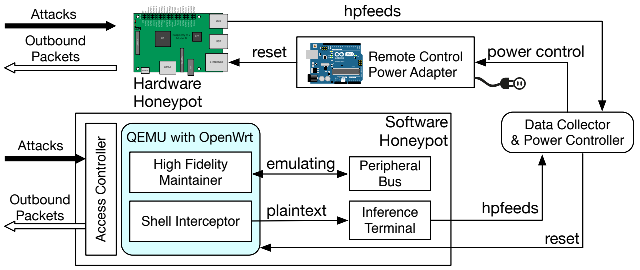


Figure 4: Architectural overview of our system, as well as the internal structure of a software IoT honeypot.

software IoT honeypot can emulate the following typical IoT devices featured by their heterogeneous architectures and compositions:

Intel Galileo Gen 1 with x86; Dreambox 600 PVR with PowerPC; BeagleBoard with ARM Cortex-A8; Orange Pi Zero with ARM Cortex-A7; Omega2 with MIPS 24K; RouterBOARD RB953GS with MIPS 74Kc.

2.3.1 *High Fidelity Maintainer.*

The High Fidelity Maintainer implements a set of strategies to prevent attackers from identifying our honeypots.

Customizing QEMU configurations. To enhance the fidelity of software honeypots, we tune the hardware configurations of QEMU in each software honeypot so that it can resemble its emulated IoT device in capability. As QEMU provides a series of CPU profiles, we select the CPU profile that best matches the CPU metrics of the emulated IoT device. We set its memory capacity as equal to that of the emulated IoT device. As we use *initramfs* to achieve an in-memory filesystem (§2.2), there is no need to emulate disks (most IoT devices do not own disks).

Masking sensitive system information. Since attackers can probe whether an IoT device is emulated by a VM based on system and kernel information (*e.g.*, by checking `/proc`) [2], we mask the VM system and kernel information. For each software honeypot, we forge `/proc/cpuinfo` in OpenWrt and make it look like a commercial CPU used by real IoT devices.

VM instances rearrangement among public clouds. Since we deploy our software IoT honeypots in public clouds, it is possible for attackers to infer our deployment based on IP addresses, because the IP ranges of public clouds can be fully or partially retrieved. For example, AWS fully releases its IP address ranges through public web APIs[8], and the IP address ranges of some public clouds can be partially acquired by examining their ASes (autonomous systems).

We built two-fold approaches to mitigate the possibility of identifying VM-based honeypots based on IP addresses. First, when we rent a VM instance from the eight public clouds, we fragmentize the IP address selection for our honeypots by fragmentizing the selection of VMs' regions, zones, and in-zone IP ranges. Second, we notice that all the eight public clouds offer the option of *elastic* IP addresses (without extra charges). Hence, we periodically change the IP address of each software honeypot. During the first six months' deployment, we noticed that 6.7% fewer attacks are captured by an average honeypot hosted on AWS than an average honeypot hosted on other public clouds. Thus, we move some honeypots from AWS to Vultr (shown in Table 2).

(a) Deployment statistics during 06/15/2017–12/14/2017

Public Cloud	#
AWS	12
Microsoft Azure	12
Google Cloud	12
DigitalOcean	12
Linode	14
Vultr	14
Alibaba	16
Tencent	16

(b) Deployment statistics during 12/15/2017–06/14/2018

Public Cloud	#
AWS	6
Microsoft Azure	9
Google Cloud	12
DigitalOcean	12
Linode	16
Vultr	21
Alibaba	16
Tencent	16

Table 2: Deployment changes of HoneyCloud. Here "#" denotes the number of deployed software honeypots.

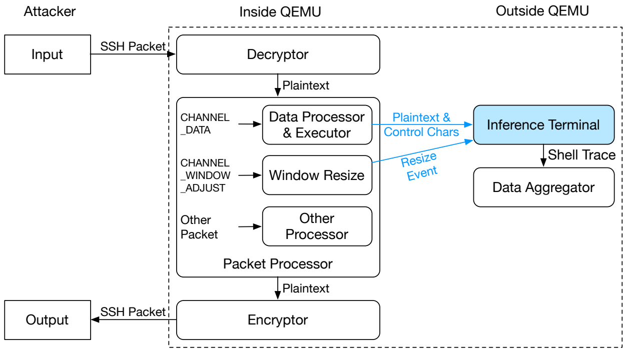


Figure 5: Working flow chart of the Shell Interceptor.

2.3.2 *Shell Interceptor & Inference Terminal.*

We build two modules, Shell Interceptor and Inference Terminal, inside and outside QEMU respectively (see Figure 4), in order to capture the actions the attackers conduct as well as the context of these actions such as operation ordering.

We modified Dropbear to recover the whole interaction process of SSH sessions. Specifically, we track the following events: connecting, logging in, resizing the window, exchanging data, and logging out.

Shell Interceptor. Figure 5 details how we extract and parse the plaintext data from the interaction traffic flows in an SSH session. When a data packet arrives at the server (using our modified Dropbear), it is first decrypted to plaintext. Next, the packet processor detects the packet type, among which we focus on `CHANNEL_DATA` (the actual terminal data) and `CHANNEL_WINDOW_ADJUST` (the resize event coming from the terminal emulator). The plaintext terminal data (including both ordinary and control characters) and the resize event are then sent to the Inference Terminal for further analysis. Here we omit the description of Telnet data interception since it is much simpler than SSH data interception.

[8] https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html

Table 3: Special escape sequences.

Sequence	Effect	Converted Text
1Bh c	Reset Terminal	[ESC c]
1Bh [H	Reset Cursor	[ESC [H]
1Bh [n J	Erase in Display	[ESC [n J]
1Bh [n K	Erase in Line	[ESC [n K]

Inference Terminal. Although the Shell Interceptor has acquired plaintext data, there are still control characters and escape sequences to be handled in the data. For example, the input sequence {"a", "b", "←", "c"} should result in "acb". If we simply ignore the control characters and escape sequences, we will get the wrong result "abc". Thereby, we need to recover the *context* of interactions. To fulfill this, we feed the shell and terminal data into *pyte*[9], a VTxxx (the video terminal standard for terminal emulators)-compatible terminal emulator. Since the terminal is screen-based (new content flushes old content), we modify the program of *pyte* to recover the full history of interactions instead of getting the interactions screen by screen.

We convert special control characters and escape sequences listed in Table 3 to plaintext so that we can know what has exactly happened. Also, we store a separate copy of keystrokes for possible hidden inputs (*e.g.,* passwords).

We track the resize event for two reasons. First, the window size of the terminal is crucial for recovering the characters that may cross lines. Second, it is a bellwether that indicates a human attacker rather than an automatic script (*i.e.,* the authenticity of the attacker).

Evidence collection. In order to collect as much information as we can for further investigation and forensics, we also report and record the following information:

- *CPU usage* as an important indicator of the execution of complex computations. For example, new types of cyber attacks, such as crypto-currency mining and ransomware, can be identified based on this information.
- *Process list* which can track any unintentional, suspicious process that indicates potential threats.
- *Network packets.* We first use *libpcap* to capture almost the full trace of network activities. Then, to filter out irrelevant packets, we ignore 1) the packets sent to the Inference Terminal and 2) the SSH packets between attackers and honeypots since these packets will eventually be handled by the Shell Interceptor.

2.3.3 Access Controller. Once a software honeypot is compromised by attackers, we should ensure that the attackers cannot utilize it to attack more IoT devices. Otherwise, HoneyCloud would become a new malware incubator. An intuitive defense is to use access control. However, the access control policy is difficult to specify in our scenario, because we do not know which outbound packets should be allowed or denied. Too restrictive policies could block benign requests (*e.g.,* DNS packets), while relaxed policies would be of high risk.

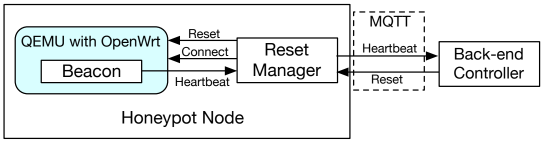


Figure 6: Heartbeat-based failure recovery.

We adopt a data-driven approach that leverage the continuously observed data to dynamically infer what packets are likely to be malicious, and then only block those dangerous outbound accesses. Our implementation employs Snort[10] (running on the host VM instance rather than the emulated IoT device) to monitor ingoing and outgoing packet traffic. By analyzing the reports generated by Snort, we can easily figure out the symmetry of data flows—asymmetry of data flows is taken as the indicator of potential attacks blocked by public clouds.

2.3.4 Reset Manager. We built *heartbeat-based reset* to periodically send heartbeat messages to the back-end controller. Once the back-end controller finds three consecutively missed heartbeats, it would send a reset command to the QEMU. We also notice that some malware (*e.g.,* Mirai) can kill the process of the SSH/Telnet server; therefore, the Reset Manager also periodically tries to connect to the QEMU via SSH/Telnet. Once the Reset Manager finds that the connection fails, it also resets the QEMU. Figure 6 demonstrates how the Reset Manager interacts with our Backend Controller.

3 FINDINGS AND IMPLICATIONS

This section analyzes the collected results of our deployed honeypots in a whole year (06/2017–06/2018). We first introduce the working flows and general statistics of our captured attacks (§3.1). Then, we present the detailed study on fileless attacks (§3.3 and §3.4), and suggest defense strategies against fileless attacks (§3.5).

3.1 General Characteristics and Statistics

As mentioned in §1, attacks on IoT devices are either malware-based or do not involve malware (fileless). The former must download certain malware files from the Internet to launch the attacks, while the latter do not need to download any malware file. Based on the attacks captured by us, their general working flows are depicted in Figure 7. For malware-based attacks, there are generally three steps to go through 1) intrusion, 2) infection, and 3) monetization. In the intrusion phase, most malware uses brute-force methods (*e.g.,* common usernames and weak passwords) to attempt to log in to IoT devices. After the attackers successfully log in, they enter the infection phase where *wget* and *tftp* are typically used to download the required malware. The downloaded malware usually connects to the command and control (*a.k.a.,* C&C) server to wait for the command. Once receiving the command, the malware can perform various kinds of attacks, such as DoS attacks and Telnet/SSH scans. Note that the malware-based attack flows we captured are basically consistent with those found in previous work [27].

For fileless attacks, the attack behaviors we captured differ from those for malware-based attacks in the second and third phases. In

[9] https://github.com/selectel/pyte

[10] https://www.snort.org/

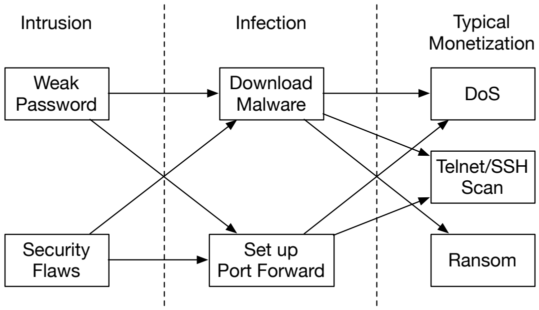


Figure 7: General working flows of captured attacks.

the infection phase, no malware is downloaded; therefore, traditional detection techniques based on malware fingerprinting cannot be applied. Instead, attackers modify certain system-level files and/or set up port forwarding and backdoors for later use. In the monetization phase, apart from DoS and Telnet/SSH scan, some other goals are achieved as detailed in §3.3 and Figure 8.

Quantitatively, from Jun. 2017 to Jun. 2018 our hardware IoT honeypots attracted more than 14.5 million suspicious connections. Among these connections, 85.8% are SSH/Telnet (ports 22, 23, 2222, and 2323) connections, 6.2% are SMB (port 445) connections, and 2.5% are HTTP(S) (ports 80, 443, and 8080) connections. As mentioned in §2, these connections led to 1.6 million effective attacks[11], where 0.75 million are malware-based attacks, and 0.08 million are fileless attacks. In other words, each hardware honeypot receives 9,930 suspicious connections and 1,100 effective attacks per day on average, showing that *attacks on IoT devices are actually frequent at present.*

Besides the four hardware honeypots, we have deployed 108 software honeypots in eight public clouds since Jun. 2017. They are geo-distributed across Europe, Asia, North America, South America, and Australia. As of Jun. 2018, we had observed 249 million suspicious connections to them, among which 78.3% are SSH/Telnet connections, 8.9% are SMB connections, and 3.2% are HTTP(S) connections. Other connections use ports 3389 (RDP), 3306 (MySQL), 1433 (SQL Server), and so on. As mentioned in §2, these connections led to 26.4 million effective attacks, where 14.6 million are malware-based attacks, and 1.4 million are fileless attacks. Meanwhile, only 37 window resize events are captured, indicating that most attacks are launched automatically (refer to §2.3.2). In other words, each software honeypot receives 6300 suspicious connections and 670 effective attacks per day on average.

Compared to an average hardware honeypot, an average software honeypot attracts 37% fewer suspicious connections and 39% fewer attacks (more specifically, 28% fewer malware attacks and 37% fewer fileless attacks). This indicates that our developed software honeypots do not possess perfect fidelity, and the reasons are mainly two folds. First, some public clouds are likely to have prevented certain types of attacks before these attacks reach our software honeypots. Second, even if attacks reach our honeypots, it is possible that some attackers leveraged more in-depth information (*e.g.*, CPU bugs and model-specific registers [29]) and advanced techniques (*e.g.*, execution analysis [9]) to infer the VM identity of our honeypots, although multi-fold endeavors have been made to mitigate the exposure of VM identity in HoneyCloud (as specified in §2.3.1). Both reasons might reduce the amount of our detectable malware and fileless attacks.

In order to understand to what extent a public cloud has carried out such prevention, we launched various common attacks (such as port scanning and brute-force authentication cracking) against our honeypots in the eight public clouds from universities and the public clouds. The attacks were launched once per hour and lasted for a whole day. The results show that all our attacks can reach our honeypots in the public clouds except Alibaba Cloud[12]. This indicates that attack filtering in public clouds may indeed be one of the possible causes, but the impact to our study is not significant since the majority (7) of the 8 public clouds we used are not affected. Besides, according to the results shown later, all the types of attacks captured by our hardware honeypots are also captured by our software honeypots (but not vice versa).

3.2 Malware-based Attacks

In total, our hardware honeypots have collected 426 different types of malware (in terms of fingerprints) downloaded by the attackers, and they can be classified into seven general categories as shown in Figure 8a according to VirusTotal. In comparison, our software honeypots have collected 598 different types of malware, fully covering the 426 different types captured by hardware honeypots. The 598 different types of malware can be classified into nine general categories as shown in Figure 8b. Among these categories, Mirai takes a significant portion—when malware binaries successfully intrude into our honeypots, 73.3% and 80.2% of them are Mirai as for hardware honeypots and software honeypots, respectively.

In addition, we find that the vast majority (~92.1%) of malware-based attacks are targeting multiple architectures of IoT devices. This characteristic is observed before by previous work [27], and we are able to confirm and further quantify it. Specifically, our collected malware (*e.g.*, Mirai, Dofloo, and Ganiw) is usually capable of running upon various architectures with various versions of binaries, including 32/64-bit x86 (28.9%), ARM (v7 & v8, 27.3% in total), MIPS [13] (Big & Little Endian, 25.7% in total), PowerPC [14] (9.2%), and SPARC [15] (8.9%). This highlights the pressing need for a flexible, software IoT honeypot solution, as a software honeypot can be easily configured to support the above architectures.

3.3 Fileless Attack Taxonomy

Besides malware-based attacks, our software honeypots have captured eight different types of fileless attacks (in terms of behaviors and intents). In comparison, our hardware honeypots have captured only five types (II - V, VII) among the abovementioned eight

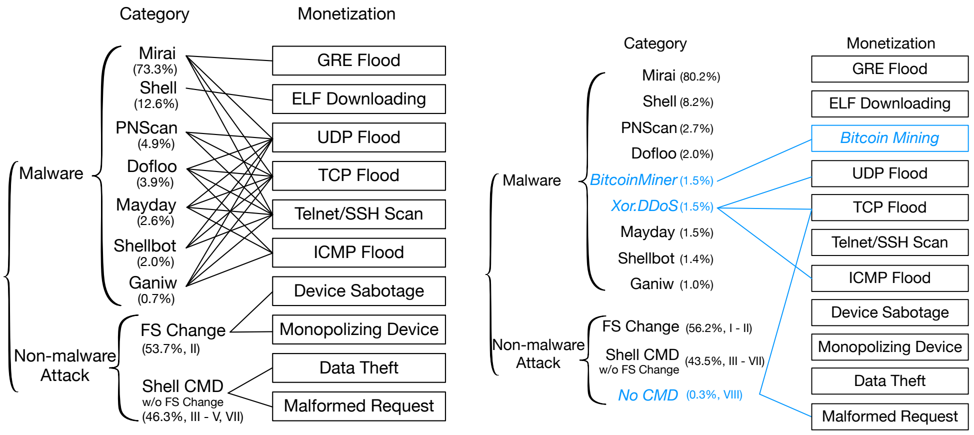
[11] A successful login is counted as an effective attack. If any file is downloaded (via `wget`), it is counted as a malware-based attack. If no file is downloaded but any command is executed, it is counted as a fileless attack.

[12] After performing port scanning using *nmap*, Alibaba Cloud blocked our SSH connection from the same machine.

[13] Broadcom and Atheros have made lots of MIPS SoCs for WiFi routers. Ingenic also provides various MIPS solutions.

[14] There are a number of PowerPC-based set-top boxes and game consoles like Wii and PlayStation.

[15] SPARC-based SoCs have been produced over years, such as LEON.



(a) Attacks captured by hardware honeypots. **(b) Attacks captured by software honeypots.**

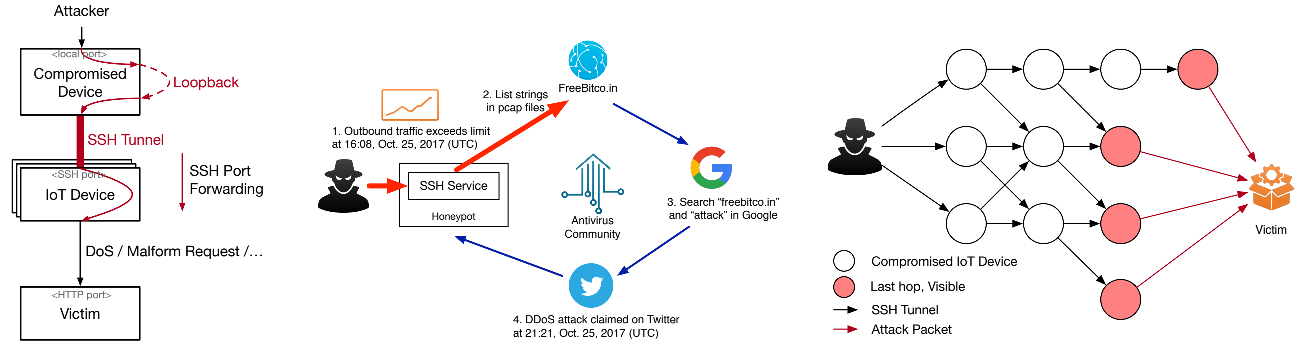
Figure 8: Attacks captured by our hardware and software honeypots in HoneyCloud. FS is short for filesystem and CMD is short for command. Note that in Figure 8b we only plot the new connection lines relative to those in Figure 8a.

types, probably because the number of hardware honeypots is much smaller. Without the malware files, we find that it is more difficult to identify and classify fileless attacks. To the best of our knowledge, in this paper we are the first to systematically characterize fileless attacks on IoT systems in the wild. Specifically, by comprehensively correlating the disclosed shell commands, the monitored filesystem change, the recorded data-flow traffic, and third parties' online reports, we identify 1.5 million fileless attacks and empirically infer their behaviors/intents as follows (the value in the parentheses denotes the percentage of the corresponding fileless attacks):

- Type I: Occupying end systems (1.8%), *e.g.,* by altering the password of an IoT device (via `passwd`). Once the password is changed, the attacker is able to access the device later and prevent other people from logging in.
- Type II: Damaging system data (54.4%), *e.g.,* by removing or altering certain configuration files or programs (via `rm` and `dd`). The typical scenario is to remove the watchdog daemon, which opens the watchdog device (`/dev/watchdog`) and makes several tests to check the system status. Once the daemon is removed, the watchdog device will not be opened. Then, the device will not reboot when it malfunctions. Thereby, the attacker may occupy and utilize the system for a longer time.
- Type III: Preventing system monitoring/auditing services (8.5%), *e.g.,* by killing the watchdog processes or stopping certain services (via `kill` and `service`). For instance, after stopping the firewall service, attackers can better exploit known vulnerabilities to launch attacks.
- Type IV: Retrieving system information (7.4%), *e.g.,* by getting the hardware information (via `lscpu`) and the system information (via `uname`, `netstat`, `ps`, and `free`). Such information may be useful for launching further attacks for specific purposes, *e.g.,* downloading and executing platform-specific malware binaries.
- Type V: Stealing valuable data (23.5%), *e.g.,* by reading passwords and/or certain configuration files (via `cat`). Note that although passwords stored in `/etc/shadow` are salted and hashed, the attackers may still be able to analyze user behaviors or recover the passwords using tools like John the Ripper password cracker[16].
- Type VI: Launching network attacks (0.3%), *e.g.,* by sending malformed HTTP requests to exploit the vulnerabilities of targeted web servers (via `wget` and `curl`) to launch DoS attacks [8]. Other typical attacks include OpenSSL Heartbleed and SQL injections.
- Type VII: Issuing other shell commands for unclear reasons (3.8%). There are other shell commands issued in our observed fileless attacks, whose purposes are currently not entirely clear to us, including `who`, `help`, `lastlog`, `sleep`, and so on. For some commands like `who` and `lastlog`, we speculate that the attacker may be collecting and analyzing other system users atop the same device.
- Type VIII: Conducting attacks where no shell command is involved (0.3%). A typical example, referred to as *SSH Tunneling Attack*, is demonstrated in Figure 9a, where the attacker has compromised an IoT device when launching this attack. Once he gets the correct credentials of the SSH service (usually via brute force) on another IoT device, he converts this (another) IoT device into a proxy server by setting up SSH port forwarding on the compromised device (*i.e.,* our honeypots, see the SSH Tunnel in Figure 9a). Usually, he binds the SSH port forwarding service (on the compromised device) to a loopback address to prevent other machines from connecting to the service. Then, the attacker launches the attack from the compromised device. Because the network flows are all proxied to the other IoT devices, the real IP address of the attacker is hidden and it is rather difficult to track such attacks.

The last type of fileless attacks is actually quite challenging to detect since there is no shell command issued and typically no filesystem change noticed. In order to pinpoint such attacks, we usually have to make cross-community explorations. Figure 9b

[16] https://www.openwall.com/john/



(a) Details of the SSH Tunneling Attack.

(b) Analysis procedure of the SSH Tunneling Attack.

(c) Network flow of the SSH Tunneling Attack.

Figure 9: (a) Details of the SSH Tunneling Attack, (b) the procedure describing how we identify the SSH Tunneling Attack, and (c) how an attacker can launch a SSH Tunneling Attack against a target.

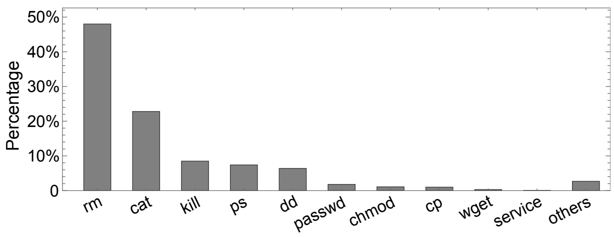


Figure 10: Usage frequency of the shell commands.

illustrates how we identify the SSH tunneling attack. First, by examining the recorded data-flow traffic, we found that the outbound network traffic in one of our deployed software honeypots suddenly exceeded a reasonable limit at 16:08, Oct. 25, 2017 (UTC). Also, the anomalous pattern of outbound traffic lasted for a couple of minutes, continuously sending HTTPS requests to a Bitcoin lottery website https://freebitco.in. Then, we searched the domain name of the website together with the keyword "attack" via Google, and fortunately, we discovered a message posted on Twitter by the website's operators, reporting that they are under DDoS attacks on the same day[17]. Furthermore, we searched for the attack in several antivirus communities (including EXETOOLS, MalwareMustDie, and IET Cyber Security Community), and found that a newly reported kind of pivoting attacks [4] well matched the behaviors of the attack we observed.[18] Also, we learned that this attack is not only stealthy but also powerful and scalable—it can easily amplify its effect by manipulating a swarm of IoT devices and establishing multiple layers of SSH tunnels (as demonstrated in Figure 9c).

3.4 Key Insights for Fileless Attacks

Aggregating the detailed study results in §3.3, we acquire several valuable insights with respect to fileless IoT attacks. First, almost all the previous studies on IoT attacks are focused on malware-based attacks, while fileless attacks have been scarcely reported. This may well mislead people to take fileless attacks as marginal and insignificant. On the contrary, our study reveals that fileless attacks are not only substantially more than expected—the number of fileless attacks is as large as 9.7% of that of malware-based attacks, but also capable of fulfilling a variety of functions and goals, some of which are pretty powerful and stealthy (*e.g.,* the SSH Tunneling Attack). Additionally, we find that a root cause of this lies in the weak authentication issue of today's IoT devices, *i.e.,* many widely-used IoT devices are using a weak password at present [24]. The issue makes it unprecedentedly easy for attackers to obtain remote control of IoT devices and then perform malicious actions without using malware.

Also, we observe that some system components (embodied by specific shell commands) are "favored" by fileless attacks. Figure 10 lists the top-10 frequently used shell commands by fileless attacks. Solely `rm` is used by 48% of the fileless attacks, while all the other commands (excluding top-10) are used by only 2.7% of the fileless attacks. In particular, the majority (65.7%) of our captured fileless attacks are launched through a small set of commands: `rm`, `kill`, `ps`, and `passwd`, which are enabled by default in our honeypots (and almost all real-world Linux-based IoT devices). For these commands, actually not all of them are necessary for a special-purpose IoT device. Thus, for a certain IoT device, if some of these frequently-exploited shell commands are indeed unnecessary, the manufacturers can disable them by customizing the device system so that the attack surface can be reduced.

In addition, we notice that fileless attacks can also help malware-based attacks or subsequent fileless attacks both stealthily and effectively. Although it is not always necessary for adversaries to acquire system information before attacking the system (*e.g.,* they may first download the malware binaries of various architectures together and then run them one by one [14]), to our observations, 39.4% of the fileless attacks are collecting system information or

[17] https://twitter.com/freebitco/status/923298533972652032

[18] Since the collected data were encrypted, we did not have direct, ideal evidence but actually noticed that the patterns well matched. Note that the SSH tunnel is used for forwarding data and does not involve shell interactions, so we cannot decrypt the data using the Shell Interceptor and Inference Terminal (§2.3.2).

performing de-immunization operations (*e.g.,* shut down the firewall and kill the watchdog) in order to allow more targeted and efficient follow-up attacks.

Although the attack vectors discussed in this section are also applied to general-purpose computers (PCs and servers), the attack effects are totally different. PCs and servers are usually highly protected with enhanced credentials, and thus they are significantly less impacted by the attacks studied in this paper.

3.5 New Security Challenges and Defense Directions

Based on the insights from our study, we are able to identify both new security challenges introduced by fileless attacks and new defense solution directions. First, 56.2% of fileless attacks (Type I–II) modify the filesystem of the compromised IoT devices. To resist such attacks, the device manufacturers should use a non-root user as the default system user.

Second, we find that 99.7% of fileless attacks (Type I–VII) are using shell commands, making these attacks detectable by auditing the shell command history of IoT devices. Unfortunately, in practice, many IoT devices use a read-only filesystem (*e.g.,* SquashFS[19]) with the hope of enhancing security. This mitigates malware-based attacks, but unexpectedly increases the difficulty (in persisting the shell command history) of detecting fileless attacks. However, if we make certain parts of the file system writable to enable shell commend logging, *e.g.,* using a hybrid filesystem (*e.g.,* OverlayFS[20]) or a versioning filesystem (*e.g.,* Elephant [30]), this inevitability makes it possible for malware-based attacks to download malware files. Thus, this is a fundamental trade-off between the auditability of fileless attacks and the security against malware-based attacks, which we think is a new IoT security challenge introduced by fileless attacks.

Third, we notice that 0.3% of fileless attacks (Type VIII) neither modify the filesystem nor execute any shell commands. Take the aforementioned SSH Tunneling Attack as an example, which exploits the SSH service in a number of IoT devices with weak passwords to construct a "tunnel" for launching attacks while hiding the real sources.

We suggest that any IoT device unsuited to using a *unique* strong password (*e.g.,* limited by the production and assembly process—low-end IoT devices are usually programmed before assembly, and thus the firmware of multiple devices is often identical) stop providing SSH service to the public Internet. In case that users have to manipulate the device remotely, they could utilize a VPN to get access to the device; in other words, the SSH service is always restrained to the local area network (LAN). The above defense strategies can be integrated and embodied in an actionable workflow called IoTCheck, as demonstrated in Figure 11. In order to assist manufacturers and administrators to validate and harden the security of their IoT devices effectively (with regard to fileless attacks), the IoTCheck workflow comprehensively examines their multi-fold information including the password strength, root privilege, filesystem type, shell environment, SSH access, unnecessary components, and so forth. Based on the checking results, easy-to-follow defense suggestions are then provided.

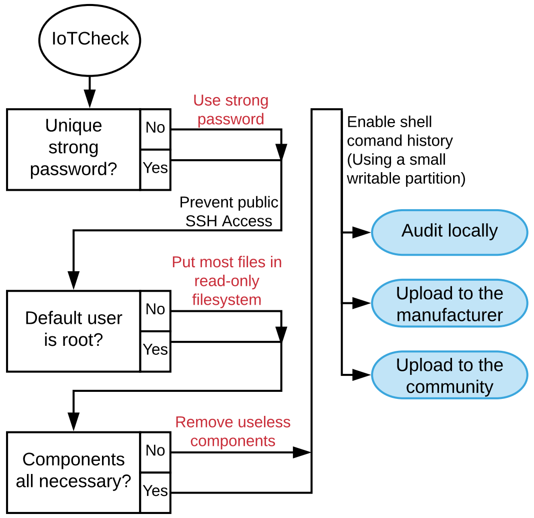

IoTCheck
Unique strong password?
No
Yes
Use strong password
Prevent public SSH Access
Default user is root?
No
Yes
Put most files in read-only filesystem
Components all necessary?
No
Yes
Remove useless components
Enable shell comand history (Using a small writable partition)
Audit locally
Upload to the manufacturer
Upload to the community

Figure 11: The IoTCheck work flow. Red texts denote that essential actions should be taken. Blue boxes represent the possible auditing schemes.

4 DISCUSSION ON LIMITATIONS

While we have comprehensively explored the practical way to build and deploy Linux-based IoT software honeypots in public clouds, still several limitations remain as to HoneyCloud.

Support of emerging IoT interfaces. Although QEMU provides versatile emulations for a wide variety of peripherals/interfaces, it still does not support some new interfaces like BT.1120 and MIPI CSI, which are typically utilized by today's IoT cameras. QEMU also lacks the support of communication interfaces like ZigBee and NFC, which are commonly seen in home automation devices. This restricts our capability to design and deploy more kinds of IoT software honeypots.

Robustness to the inference of VM identity. Multi-fold endeavors have been made to mitigate the exposure of VM identity in HoneyCloud. However, it is still possible that some attackers may leverage more in-depth information (*e.g.,* CPU bugs and model-specific registers [29]) and advanced techniques (*e.g.,* execution analysis [9]) to infer the VM identity of our software honeypots. Besides, in order to study the concrete effectiveness of these above-mentioned techniques, we may need to substantially upgrade our current experiment design.

In-depth analysis on advanced attacks. HoneyCloud currently cannot decrypt the traffic delivered in the SSH tunnel; therefore, we typically do not know what attackers actually do through the tunnel and thus are not able to analyze the intentions of SSH tunneling attacks. However, if the attackers send plaintext requests, we can

[19] https://www.kernel.org/doc/Documentation/filesystems/squashfs.txt
[20] https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt

still capture and try to understand them. In addition, Advanced Persistent Threats (APTs [32]) have been becoming popular recently. Since HoneyCloud resets the deployed honeypots every now and then and thus does not well persist attacks, HoneyCloud is currently not quite suited to the analysis of APTs.

5 RELATED WORK

Fileless attacks have gone through a long history as they are perhaps the most direct and sophisticated attacks. This section reviews prior studies on fileless attacks (§5.1) on both traditional PCs/workstations and IoT devices. We also discuss existing honeypot solutions (§5.2) and compare them with HoneyCloud.

5.1 Fileless Attacks

As early as the 1980s (before the first batch of computer malware, *e.g.,* the Morris Internet worm, emerged [22]), fileless attacks had been conducted on PCs and servers. Symantec's Internet Security Threat Report [34] categorizes security attacks into three types based on their incursion methods. First, in the *in-memory attacks*, attackers exploit vulnerabilities in the operating system or service programs [21]. A typical example is the EternalBlue attack, which exploits a vulnerability in the Windows Server Message Block (SMB) protocol to allow remotely executing arbitrary code via crafted packets [7]. Second, in the *non-PE (portable executable) file attacks*, adversaries take advantage of system tools available on the host systems (*e.g.,* PowerShell and Microsoft Word) rather than PE files to launch attacks [19]. Last but not the least, *weak or stolen credentials* are often used to intrude the victim PCs or servers.

Fileless attacks against IoT devices had not been actively reported until in recent years [10, 13]. For example, the research workgroup MalwareMustDie discovered a criminal gang using the technique of SSH TCP direct forwarding to launch fileless attacks mostly targeting HTTP(S) servers [4].

5.2 Honeypot Solutions

Ever since late 1990s, honeypots have been widely developed to capture and analyze potential attacks on PCs and commodity servers, with respect to Internet services such as remote login, email, and web services. Most of the well-known Internet honeypots have been included in the Honeynet project [20]. In general, designing a honeypot system usually needs to balance the tradeoff among four major properties: fidelity, scalability, persistence, and containment. Low-interaction honeypots (LIHs) typically simulate certain services or aspects of a networked system, and thus are easier and less resource-intensive to deploy in a large scale. Since they do not actually provide a service or execute the functionality code, their fidelity is fairly low or limited, but their persistence and containment are high. On the contrary, to keep high fidelity, high-interaction honeypots (HIHs) offer full system functionality, which is typically not supported by LIHs. This inevitably increases the difficulty and complexity to achieve scalability, persistence, and containment.

As a representative LIH, Honeyd [28] can simulate multiple computer systems at the network level in a flexible manner. Its flexibility mainly lies in its capability to simulate any common services as long as the service is pre-configured via its plug-in framework. It can also approximate the network stack of different OSes and provide arbitrary routing topologies. Honeyd is able to deceive many fingerprinting and network mapping tools, but can be easily recognized (as a honeypot rather than a real Internet server) in several inexpensive ways [26]. In addition, by automatically generating intrusion detection signatures for malicious network traffic patterns, Honeycomb [23] effectively extends the functionality of Honeyd, while does not essentially improve the fidelity and containment.

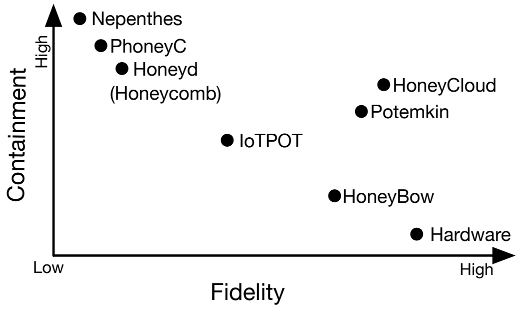


Figure 12: Design tradeoff of honeypots.

Nepenthes [12] and PhoneyC [25] are also LIHs. The former concentrates on emulating the vulnerable parts (i.e., known vulnerabilities) of services to collect the information of malware at scale, thus leading to better efficiency and scalability. The latter is distinguished by its running at the client side (most honeypots run at the server side), and thus is also called *honeyclient*. Targeting malware in the web, PhoneyC emulates a fully featured HTTP client coupled with dynamic language support and user behavior mimicking.

Vrable *et al.* built an HIH farm named Potemkin [33]. While higher interactions often imply less scalability, Potemkin achieves good scalability by leveraging operating system virtualization, aggressive memory sharing, and late binding of resources. HoneyBow [38] is a high-interaction honeypot dedicated to collecting the information of malware. Different from Nepenthes, HoneyBow closely monitors the file system and cross-checks the file list, instead of analyzing the network flows.

There have been a number of studies on IoT attacks using honeypots. For example, based on the deployment of their developed IoTPOT systems, Minn *et al.* discover that Telnet-based attacks against IoT devices have rocketed since 2014 [27]. Moreover, Gandhi *et al.* set up a proxy-like honeypot system named HIoTPOT to separate malicious users from authenticated users and store the detailed information of malicious users [15]. Yang *et al.* set up hardware honeypots for malware forensics [35]. Nevertheless, none of the above efforts have ever captured or reported fileless attacks.

Figure 12 roughly positions the abovementioned systems (as well as our HoneyCloud) in the design space of honeypots. We only illustrate fidelity and containment as the two fundamental dimensions. Compared with traditional honeypot designs such as Potemkin, HoneyCloud has higher fidelity with the efforts of High Fidelity Maintainer described in §2.3.1 (the hardware honeypots have the highest fidelity). It also achieves higher level of

[21] Attackers can also pursue persistence of fileless attacks, *e.g.,* by storing a malicious script in the registry of a Windows PC, employing the scheduled task mechanism, or leveraging the Windows Management Instrumentation (WMI) [17].

containment with the access control mechanism described in §2.3.3, compared with other HIHs such as Potemkin, IoTPOT, and HoneyBow. Certainly, LIHs including Nepenthes, PhoneyC, and Honeyd achieve the highest level of containment by giving away fidelity. Our results and real-world deployment validate that HoneyCloud makes good design and implementation tradeoffs and is able to attract both malware-based and fileless attacks.

6 CONCLUSION

IoT attacks are pervasive and becoming increasingly severe as security threats. Existing researches in addressing IoT attacks mainly focus on malware-based IoT attacks. While malware-based attacks (*e.g.,* Mirai) can quickly spread across IoT devices, they can be effectively resisted by malware fingerprinting or static/dynamic malware analyses. In the past few years, fileless attacks have been increasingly observed and reported, in particular on Linux-based IoT devices. Such attacks pose significant threats to the widely deployed IoT devices across the world, but have not been systematically studied or comprehensively analyzed. To understand Linux-based IoT attacks with a focus on fileless attacks, we build and deploy several IoT devices as hardware honeypots, and a large number of specially designed software honeypots in multiple public clouds, to capture and analyze such attacks at scale in the wild. We collect the data of a variety of real-world fileless attacks, profile their characteristics and influences, and propose actionable defense strategies. We believe that traditional threat models of IoT security, which have been focusing on malware, need to take fileless attacks into serious consideration, and a global, unified defense framework for fileless attacks is in dire need.

ACKNOWLEDGMENTS

We sincerely thank our shepherd Dr. Urs Hengartner and the anonymous reviewers for their valuable feedback. We also appreciate Yongfeng Zhang, Jie Li, Chen Geng, and Jinsong Ma for their contributions to the deployment of HoneyCloud. This work is supported in part by the National Key R&D Program of China under grant 2018YFB1004700, the National Natural Science Foundation of China (NSFC) under grants 61632013, 61822205, 61432002 and 61632020, and the Key R&D Program of Zhejiang Province under grant 2018C01088.

REFERENCES

[1] Internet of Things Security Research Report, 2017. http://www.nsfocus.com.cn/upload/contents/2017/12/20171205171653_35944.pdf. (Accessed on Mar. 15, 2019).

[2] Linux - Easy Way to Determine Virtualization Technology - Unix & Linux Stack Exchange. https://unix.stackexchange.com/questions/89714/easy-way-to-determine-virtualization-technology. (Accessed on Dec. 26, 2017).

[3] McAfee Labs: Cybercriminal Tactics Shifting From External Malware Threats to 'fileless' Attacks. https://www.dqindia.com/mcafee-labs-cybercriminal-tactics-shifting-external-malware-threats-fileless-attacks/. (Accessed on Dec. 13, 2018).

[4] MMD-0062-2017 - Credential Harvesting by SSH Direct TCP Forward Attack via IoT Botnet. http://blog.malwaremustdie.org/2017/02/mmd-0062-2017-ssh-direct-tcp-forward-attack.html. (Accessed on Dec. 26, 2017).

[5] New Trends in the World of IoT Threats. https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/. (Accessed on Mar. 15, 2019).

[6] Now You See Me: Exposing Fileless Malware – Microsoft Secure. https://cloudblogs.microsoft.com/microsoftsecure/2018/01/24/now-you-see-me-exposing-fileless-malware/. (Accessed on Sep. 01, 2018).

[7] NVD - CVE-2017-0143. https://nvd.nist.gov/vuln/detail/CVE-2017-0143. (Accessed on Sep. 18, 2018).

[8] NVD - CVE-2018-7262. https://nvd.nist.gov/vuln/detail/CVE-2018-7262. (Accessed on Sep. 11, 2018).

[9] QEMU Emulation Detection. https://wiki.koeln.ccc.de/images/d/d5/Openchaos_qemudetect.pdf. (Accessed on Dec. 26, 2017).

[10] Tips for Guarding Against Untraceable, "Fileless" Cyberattacks. http://www.govtech.com/security/Tips-for-Guarding-Against-Untraceable-Fileless-Cyberattacks.html. (Accessed on Sep. 18, 2018).

[11] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, et al. 2017. Understanding the Mirai Botnet. In *Proceedings of USENIX Security*. Vancouver, BC, Canada.

[12] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. 2006. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *Proceedings of USENIX RAID*. Hamburg, Germany.

[13] Fan Dang, Ennan Zhai, Zhenhua Li, Pengfei Zhou, Aziz Mohaisen, et al. 2019. Pricing Data Tampering in Automated Fare Collection with NFC-Equipped Smartphones. *IEEE Transactions on Mobile Computing* 18, 5 (May 2019), 1159–1173.

[14] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi. 2018. DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation. *Security and Communication Networks* (Feb. 2018), 7178164:1–7178164:30.

[15] Usha Devi Gandhi, Priyan Malarvizhi Kumar, R. Varatharajan, Gunasekaran Manogaran, Revathi Sundarasekar, et al. 2018. HIoTPOT: Surveillance on IoT Devices against Recent Threats. *Wireless Personal Communications* (Jan. 2018).

[16] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. 2014. Malware Analysis and Classification: A Survey. *Journal of Information Security* 05 (Jan. 2014), 56–64.

[17] Matt Graeber. 2015. Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asyncronous, and Fileless Backdoor. In *Black Hat*. Las Vegas, NV, USA.

[18] Jun Han, Shijia Pan, Manal Kumar Sinha, Hae Young Noh, Pei Zhang, et al. 2018. Smart Home Occupant Identification via Sensor Fusion Across On-Object Devices. *ACM Transactions on Sensor Networks* 14, 3-4 (Dec. 2018), 23:1–23:22.

[19] Danny Hendler, Shay Kels, and Amir Rubin. 2018. Detecting Malicious PowerShell Commands Using Deep Neural Networks. In *Proceedings of ACM ASIACCS*. Incheon, Republic of Korea.

[20] The Honeynet Project. http://www.honeynet.org/. (Accessed on Dec. 26, 2017).

[21] ISO/IEC 20922:2016 Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1. http://www.iso.org.

[22] Brendan P. Kehoe. 1992. *Zen and the Art of the Internet*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[23] Christian Kreibich and Jon Crowcroft. 2004. Honeycomb: Creating Intrusion Detection Signatures Using Honeypots. *SIGCOMM Computer Communication Review* 34, 1 (Jan. 2004), 51–56.

[24] Franco Loi, Arunan Sivanathan, Hassan Habibi Gharakheili, Adam Radford, and Vijay Sivaraman. 2017. Systematically Evaluating Security and Privacy for Consumer IoT Devices. In *Proceedings of ACM IoT S&P*. Dallas, Texas, USA.

[25] Jose Nazario. 2009. PhoneyC: A Virtual Client Honeypot. In *Proceedings of USENIX LEET*. Boston, MA, USA.

[26] Jon Oberheide and Manish Karir. 2010. Honeyd Detection via Packet Fragmentation. (Jul. 2010).

[27] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, et al. 2015. IoTPOT: Analysing the Rise of IoT Compromises. In *Proceedings of USENIX WOOT*. Washington, D.C., USA.

[28] Niels Provos. 2004. A Virtual Honeypot Framework. In *Proceedings of USENIX Security*. San Diego, CA, USA.

[29] Thomas Raffetseder, Christopher Kruegel, and Engin Kirda. 2007. Detecting System Emulators. In *Proceedings of ISC*. Valparaíso, Chile.

[30] Douglas J. Santry, Michael J. Feeley, Norman C. Hutchinson, and Alistair C. Veitch. 1999. Elephant: The File System That Never Forgets. In *Proceedings of ACM HotOS*. Rio Rico, AZ, USA.

[31] Lance Spitzner. 2003. *Honeypots: Tracking Hackers*. Vol. 1. Addison-Wesley Reading.

[32] Colin Tankard. 2011. Advanced Persistent Threats and How to Monitor and Deter Them. *Network Security* 2011, 8 (Aug. 2011), 16 – 19.

[33] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, et al. 2005. Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm. In *Proceedings of USENIX SOSP*. Brighton, United Kingdom.

[34] Candid Wueest and Himanshu Anand. 2017. *Living Off the Land and Fileless Attack Techniques*. Technical Report.

[35] Jingyu Yang and Fan Dang. 2017. An IoT Honeypot Device for Malware Forensics. In *AVAR*. Beijing, China.

[36] Zimu Zhou, Chenshu Wu, Zheng Yang, and Yunhao Liu. 2015. Sensorless Sensing with WiFi. *Tsinghua Science and Technology* 20, 1 (Feb. 2015), 1–6.

[37] Tong Zhu, Qiang Ma, Shanfeng Zhang, and Yunhao Liu. 2014. Context-free Attacks Using Keyboard Acoustic Emanations. In *Proceedings of ACM CCS*. Scottsdale, AZ, USA.

[38] Jian-wei Zhuge, Xin-hui Han, Yong-lin Zhou, Cheng-yu Song, Jin-peng Guo, et al. 2007. HoneyBow: An Automated Malware Collection Tool based on the High-Interaction Honeypot Principle. *Journal of China Institute of Communications* 28, 12 (Dec. 2007), 8.