

Honware: A Virtual Honeytrap Framework for Capturing CPE and IoT Zero Days

Alexander Vetterl

Computer Laboratory, University of Cambridge
Cambridge, UK
alexander.vetterl@cl.cam.ac.uk

Richard Clayton

Computer Laboratory, University of Cambridge
Cambridge, UK
richard.clayton@cl.cam.ac.uk

Abstract—Existing solutions are ineffective in detecting zero day exploits targeting Customer Premise Equipment (CPE) and Internet of Things (IoT) devices. We present honware, a high-interaction honeypot framework which can emulate a wide range of devices without any access to the manufacturers’ hardware. Honware automatically processes a standard firmware image (as is commonly provided for updates), customises the filesystem and runs the system with a special pre-built Linux kernel. It then logs attacker traffic and records which of their actions led to a compromise. We provide an extensive evaluation and show that our framework improves upon existing emulation strategies which are limited in their scalability, and that it is significantly better both in providing network functionality and in emulating the devices’ firmware applications – a crucial aspect as vulnerabilities are frequently exploited by attackers in ‘front-end’ functionalities such as web interfaces. Honware’s design precludes most honeypot fingerprinting attacks, and as its performance is comparable to that of real devices, fingerprinting with timing attacks can be made far from trivial. We provide four case studies in which we demonstrate that honware is capable of rapid deployment to capture the exact details of attacks along with malware samples. In particular we identified a previously unknown attack in which the default DNS for an ipTIME N604R wireless router was changed. We believe that honware is a major contribution towards re-balancing the economics of attackers and defenders by reducing the period in which attackers can exploit zero days at Internet scale.

I. INTRODUCTION

The Internet is transforming from an Internet of computers to a global network connecting everyday devices (‘things’). The emphasis on automation and the nature of the devices themselves together mean that vulnerabilities and exploits not affecting device functionality are likely to remain unnoticed by their owners. Recent Distributed Denial of Service (DDoS) attacks which used inadequately secured Customer Premise Equipment (CPE) and Internet of Things (IoT) devices [1], [2], [3] further highlight that existing defenses are slow to detect zero day exploits and capture attack traffic. This means that attackers have considerable periods of time to find and compromise vulnerable devices before the attack vectors are well understood and subsequent mitigation is in place [4].

Honeypots, resources that appear to be legitimate systems, have long proven effective in capturing malware, helping to counter spam and providing early warning signals about upcoming threats [5], [6], [7], [8]. The Mirai botnet, the first

large botnet to recruit a wide variety of IoT devices, used an automated pseudo-random scanning process to find and infect new devices. As the Mirai source code has been leaked and is well-understood, it is straightforward to build a honeypot that emulates a vulnerable device by sending appropriate strings back to scanners. But, without source code, or reverse engineering the malware binaries, this type of honeypot is hard to construct. It is a significant challenge to monitor large numbers of attackers who are going after a wide range of devices using different attack techniques, some of which may be previously unknown ‘zero days’.

Meanwhile, it has become feasible to scan the whole IPv4 address space for vulnerable devices with modest investment. Tools such as *Shodan* or *ZMap* [9] give attackers a crucial advantage. Once an exploit is found for one technology, device, or specific implementation, attackers can easily find devices with that vulnerability embedded – and instantly benefit from that exploit. In 2003 Spitzner argued that honeypots “get little traffic” and “collect small amounts of high-value data” [10]. However, this observation was from a time when attacks were generally performed by humans, whereas since the rise of botnets almost all activities that honeypots observe are performed by automated scripts.

Previous research includes Firmadyne [11], an analysis system that runs embedded firmware and subsequently provides dynamic analysis capabilities, and IoT POT [12], one of the first generic high-interaction honeypot tailored to impersonate IoT devices. IoT POT supports eight architectures including ARM, MIPS and X86 and aims to return appropriate strings to connections on port 23 (Telnet). If the command is unknown, it tries to run it within a generic sandboxed environment to infer the appropriate return string(s). In 2017, Guarnizo et al. [13] presented a “scalable high-interaction” honeypot platform called SIPHON which is based on physical devices. They exposed six security cameras, one networked video recorder and one networked printer through a distributed architecture on a range of IPv4 addresses.

All these approaches have critical shortcomings: Firmadyne is built for dynamic analysis, but not to monitor a large number of attackers and thus not to be connected to the Internet. IoT POT does not use firmware images of real devices and thus it is a generic representation of a vulnerable platform which will fail to detect new attack patterns; SIPHON needs

physical devices connected to the Internet to capture attack traffic – an expensive endeavour limited in its scalability.

We present honware, the first flexible and generic framework to efficiently and effectively deploy honeypots for networked devices on the Internet to log attacker traffic and their actions. Instead of buying CPE or IoT devices and running them as honeypots, honware utilises device firmware images (which are widely available for download) and a special pre-built Linux kernel to emulate device behaviour within a virtual environment. In particular, it is easy to deploy all the available firmware versions for a particular device so as to understand which are vulnerable to a particular attack.

Overall, we make four main contributions:

- We present new techniques (and improve upon existing work) to allow honware to run standard firmware images without needing custom hardware.
- We show that honware is superior to existing emulation strategies, making significant improvements in scalability, in the provision of network functionality, and in emulating firmware applications.
- We perform extensive measurements to show that the performance of honware is comparable to real devices and that honware is not susceptible to trivial fingerprinting based on timing attacks.
- We present four examples to show the success of honware in identifying real-world attacks which had been hard to capture with the traditional approach of low-/medium-/high- interaction honeypots.

II. BACKGROUND

Honeypots are classified by the type of system they emulate, such as SSH, web or email servers. They are further classified as low interaction (in the context of SSH merely collecting credential guesses), medium interaction (emulating a limited number of shell commands) or high interaction (allowing attackers full control of a machine). Low- and medium-interaction honeypots are easy to deploy and maintain but typically only implement a small subset of system features. In contrast, high-interaction honeypots expose a complete system and so any vulnerabilities can be exploited, whether or not they were previously known. Once defenders fully understand a new attack, a custom low or medium interaction honeypot can be built to capture that specific attack traffic and provide quantitative data about the level of harm along with operational data such as current malware samples and C&C locations.

Since mid-2016, and the rise of the Mirai botnet, there has been an increasing interest in ‘customer-premise equipment’ (CPE), a generic term for xDSL modems, routers, switches and other home networking devices that are connected to telecommunication providers’ networks. CPE such as ADSL and cable modems are particularly important within the IoT ecosystem as they serve as an entry point into a premises’ network and, once compromised, allow attackers to connect to further devices at that location. The firmware of CPE and IoT devices is largely based on Linux, typically supplemented with custom kernel modules and drivers to provide device specific

functionality. Firmware for CPEs is almost invariably available for download and Netgear [14], Link-TP [15] and Linksys [16] provide online GUI emulators for various types of modems and routers. An increasing number of manufacturers also publish the source code of their firmware as GPL-Code, in particular to support ongoing projects such as OpenWrt [17].

In the arms race with the criminals, honeypot ‘fingerprinting’ is an ongoing concern. Morishita et al. [18] showed that many honeypots do not blend into the type of service they emulate and reveal themselves based on their unique configuration. Vetterl and Clayton [19] developed an automated technique to fingerprint honeypots based on packet level protocol interactions, identifying honeypots with trivial probes. Sophisticated attackers who use fingerprinting to avoid interacting with honeypots could avoid detection for long periods, but honware sidesteps all these issues by running exactly the same applications and protocol stack as a real device.

It was shown by Garfinkel et al. [20] that virtualisation induces anomalies such as timing discrepancies and that these anomalies can be used to detect virtualised environments. Kedrowitsch et al. [21] compared the deception capabilities of Linux containers with different virtualisation environments such as QEMU, Kernel-based Virtual Machine (KVM) and VMWare, and found that QEMU can be best fingerprinted by its slow performance. Similarly, Holz and Raynal [22] showed that the execution time of commands provides an efficient way to detect honeypots because emulation will typically result in longer execution and response time. In the same vein, Mukkamala et al. [23] found that honeypots running in virtualised environments typically respond slower than real services. However, they also demonstrated that for Internet-connected honeypots this metric may not be useful because it depends on network load, routing and emulation technology.

III. VIRTUAL HONEYPOT FRAMEWORK

Honware uses the firmware images provided by CPE and IoT manufacturers, employing Quick Emulator (QEMU) to run code for different CPU architectures (x86-64 PCs, ARM, MIPS and PowerPC) on a single host machine. Although parts of the firmware will be closely linked to the hardware of a particular device, the Linux kernel and the device driver APIs are substantially the same across many different devices. Honware decouples the execution of the firmware from the underlying hardware by the use of a custom kernel.

Figure 1 shows the four main parts of honware: a host operating system and kernel, QEMU, a custom kernel, and the firmware filesystem (extracted from the firmware image) which contains applications such as telnet and web servers.

QEMU is the de facto standard for full machine emulation and it provides the necessary functionality to connect the honeypot to the Internet. However, QEMU cannot be used for off-the-shelf emulation of CPE and IoT firmware images. The firmware, with its own kernel, will try to communicate with the hardware, but its absence means that this communication will fail. For example, any access to non-volatile memory (nvram)

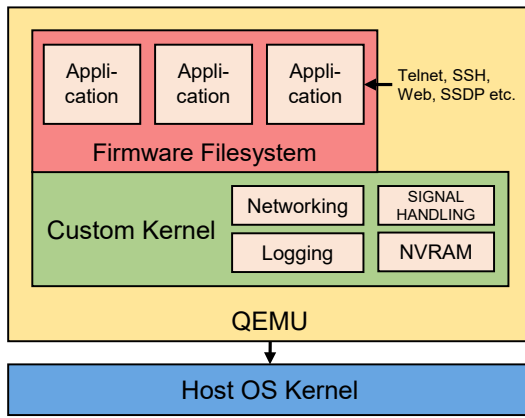


Fig. 1. Honware architecture overview: Honware consists of four main parts, a host operating system and kernel, Quick Emulator (QEMU), a custom kernel, and the firmware filesystem itself which contains specific applications such as telnet and web servers.

to read configuration files or an attempt to retrieve the MAC address from hardware will fail. Thus we run QEMU with our own customised kernel and the extracted filesystem on top of a host operating system such as FreeBSD or Linux Ubuntu.

Honware currently uses three custom kernels since these are sufficient for handling very large numbers of firmware images. These are the Linux kernels 2.6.32.70 for MIPS little endian (mipsel) and MIPS big endian (mipseb), and 4.1.52 for ARM. We purposely used older kernels as these are most prevalent across CPE and IoT firmware images and support for newer kernels is rarely required. The MIPS kernels are compiled with MIPS Malta and with the MIPS32 CPU release version 2. MIPS Malta is particularly useful as it supports the emulation of PCI and LAN functionalities that are inherently required for the emulation of networked equipment. The ARM kernel is compiled with the architecture Versatile Express and VIRT/MULTI_V6_V7 as a *dummy* CPU architecture. As the Versatile Express platform is meant for development and rapid prototyping, it provides a wide range of hardware support. We further configure the kernel to support various networking features such as VLAN and WLAN.

The emulation of firmware images is performed in three stages and is fully automated: A) extracting the filesystem from the firmware image, B) modifying the filesystem to allow for virtualisation and C) running it in QEMU with one of our pre-compiled kernels. Honware will typically process a firmware image (typically a .zip or .rar file) and have the honeypot ready to run within one minute.

A. Automated firmware extraction

Firmware images for routers, ADSL modems and IoT devices are widely available online and many manufacturers provide regular updates on their websites. Honware uses these images as input, usually in a compressed format, and extracts the firmware filesystem. Any supplied kernel is ignored as running it within QEMU would not allow us to interfere with the execution of the filesystems applications and services – a

necessity to run the firmware image as a honeypot and in fact, to decouple the firmware image from the underlying hardware.

To extract the filesystem, we use binwalk [24] and recursively look for the Linux filesystem structure. We identify Linux filesystems by determining whether a folder contains the necessary root structure including `bin`, `usr` and `proc`. This has proven to be challenging as not all firmware images are packed in the same way. Occasionally they include multiple suitable filesystems such as one firmware image for the upgrade, one for the previous version and one for a factory reset. We also found instances in which a secondary filesystem is mounted during the boot process, for example to provide scratch space. In such a case, we only consider the filesystem where the `init` process resides as without this process, the firmware will not boot at all. Although extraction is intended to be completely automatic, honware does allow manual selection of which filesystem to use should this be needed.

We then use `qemu-img` to create a 2GB raw file, subsequently create an ext2 filesystem and copy the root folder structure including all files and binaries. We infer the CPU architecture by reading the ELF header of the biggest binary of the filesystem, typically Busybox. We further extract all certificate (.pem) files from the firmware images so that we can use tools such as Wireshark to decrypt, for example, HTTPS traffic to the web server.

It is more challenging to decrypt SSH traffic as the Diffie-Hellman key exchange uses not only a static key, but also a session key. Retrieving the relevant session key is not straightforward and requires locating a particular memory structure in guest memory. We decided not to tackle this particular issue as we can log executed commands in our custom kernel. However honeypot operators could consider SSLSNOP [25] or related techniques to collect the raw traffic.

B. Customised pre-built kernel

We now consider the various components of our customised Linux kernel and explain how we have managed to improve on earlier systems such as Firmadyne. In particular, previous work has ignored the kernel’s signal handler as way of ensuring that applications continue running and do not terminate silently, and no previous attempts have been made to support out-of-tree kernel modules. Furthermore, our kernel does not solely rely on the default configuration of the guest system to ensure network connectivity, but can force a particular network configuration to be used. In addition, honware supports up to four ethernet devices for the ARM little-endian platform whereas Firmadyne only supports a single ethernet device.

1) *Honeypot logging and module loading*: In order for honeypots to provide insights into how systems are attacked and to monitor subsequent actions, it is essential to have extensive logging capabilities. To get details of all executed programs and commands with the appropriate time stamps, process ids and contexts, we modified the kernel function `do_execve`. For each connection, we create a new session id under which we log the invoked programs and their details. Commands which are not associated with network connections

are logged with the default session id 0. This means that the boot process, cron jobs and other processes that are executed by the firmware image itself can be clearly distinguished from actions triggered from the outside.

2) *Signal interception*: One central problem we encountered is that the init process and various applications frequently terminate silently or with generic error messages. Applications may terminate because of wrong or missing nvram values, incorrect hardware emulation or missing kernel features. To mitigate the effects of this problem, we modified the signal handling in the kernel to 1) not allow the kernel to terminate the process, for example by means of the default signal handler and 2) not allow the program to terminate itself with, for example, a SIGABRT signal. This means that applications and kernel modules continue their execution irrespective of what signals are sent. To achieve this we modified the kernel function `get_signal()` which is called by `do_signal()` and is responsible for signal handling in the kernel.

It has proven particularly useful to intercept SIGNAL 6 (SIGABRT). We find that SIGABRT is used to detect broken constraints, for example, to indicate missing license keys or the absence of hardware modules. In particular, devices with Broadcom chips look for a variety of settings in nvram to differentiate between hardware versions. If these settings cannot be found, the init process or the calling application wants to terminate itself to prevent further execution. In addition to SIGNAL 6, we also intercept SIGNAL 11 (SIGSEGV) and SIGNAL 7 (SIGFPE). The latter two mitigate the problems caused by missing nvram values that are not absolutely necessary for running applications. SIGFPE is typically sent for floating point errors such as when the application attempts to divide by 0. We believe that the absence of nvram leads to the use of zero values and various mathematical operations fail.

For a small number of firmware images' ignoring signals leads to indefinite loops and very high CPU usage, however in many cases the programs appear to continue running successfully. Thus intercepting signals is a trade-off between ignoring (some) values, while at the same time making the emulation useful and viable.

3) *Module loading*: A number of firmware images are shipped with custom modules to implement device specific functionality such as cryptographic operations or support for custom hardware. To avoid undesired behaviour and kernel crashes, our customised kernel will not load incompatible modules, even if instructed (`modprobe -f`). However, the kernel will accept modules with different `vermagic` strings and does not require exact matches. `vermagic` strings are typically used to ensure that the modules were built and configured for the same kernel version, but as the `vermagic` strings greatly differ between manufacturers and firmware versions, we ignore them. For example, if a module has the `vermagic` string `2.6.22-routeros` our newer kernel with the version `2.6.32.71` will attempt to load it. This problem could be avoided by re-compiling the kernel for every individual firmware image with the correct `vermagic` string, but this would be a significant barrier to deploying honeypots in

a timely manner and we find that, in practice, our approach works very well.

4) *NVRAM*: To store and later retrieve device-specific configuration parameters, device manufacturers often use non-volatile memory (nvram). Chen et al. [11] looked at 23 035 firmware images and found that more than half of them accessed nvram, for example to handle configuration information during the boot process. Typically firmware images set a variety of nvram values during the boot process and subsequently read these nvram values with `nvram_get`. To emulate nvram, we use the approach first mentioned in [26] and later developed by Firmadyne: we set the environmental variable `LD_PRELOAD` to the path of our own nvram implementation, so that our file will be loaded before any other (firmware) library. This means that we reliably intercept calls to `nvram_get` and `nvram_set`.

To extend Firmadyne's approach, we implement a script that automatically reads the kernel logs, detects missing nvram values, re-compiles the necessary shared library and updates the filesystem for the next time the firmware is run. This can be an iterative process as setting certain nvram values may cause other nvram values to be accessed and those could also be missing. In our evaluation (in Section IV) which compares honware with Firmadyne, we used static values and did not iteratively look for missing values. However, as shown in Table I and II, we achieve far better results than Firmadyne in terms of correctly emulating the firmware, its applications and in inferring the network configuration.

5) *Network configuration*: Providing network functionality is a fundamental prerequisite for honware since without it the emulation cannot be connected to the Internet and thus probed by attackers. To infer the network configuration, honware parses the kernel logs for the initial configuration of the bridge device, typically named `br0` or `ra0`. It then looks for `ifconfig` commands which configure the bridge, and for any `addif` command which adds one or multiple network interfaces to that bridge. Subsequently, it extracts the IP address and creates a tap interface in QEMU, and sets the associated route and iptables rules on the host to forward all traffic to the inferred network interface.

If the network does not work with the inferred configuration, we re-run the emulation and overwrite the firmware's default network configuration with a custom configuration. This is achieved by placing the configuration into `/sbin/boot.sh` which will be executed by the kernel during the boot process. The custom configuration starts by shutting down the network interfaces (`eth_x`) and any wrongly configured bridges. Then it assigns appropriate local static IP addresses and adds the network interfaces back onto the bridge, setting up a route so that the guest network interface and the host's network interface can exchange packets. Finally, the script removes all firewall rules so that we can be certain that traffic is not interfered with. If the network still does not respond to ICMP echo request packets, we consider the firmware image not *network reachable* (and our attempt to create a useful honeypot has been unsuccessful).

To fully test network reachability we use `ping` and `nmap`. In particular we use `nmap`¹ to do a port scan of the most common ports including port 22 (SSH), 23 (Telnet), 80 (HTTP), 443 (HTTPS) and 1900 (UPnP). The results of this evaluation with 8387 firmware images can be found in Section IV-A.

C. Filesystem modifications

After extracting the filesystem, we have to modify it for emulation. First, we add the module for nvram emulation (Section III-B4). Second, we modify `do_execve` to execute, if present, `/sbin/boot.sh` through the kernel function `call_usermodehelper`. This allows us to execute custom scripts and commands for a particular firmware image without having to change the pre-built kernel or perform complex modifications to the firmware filesystem. This gives honeypot operators flexibility to, for example, specify additional network interfaces, execute applications with particular configuration options or customise the firewall to their needs.

Unsurprisingly, many emulated firmware images do not configure static IP addresses, but use DHCP. As we choose not to emulate access to a DHCP server, attempts to obtain an IP address would fail and the emulated firmware would not be reachable over the network. To address this, we use `busybox-static`, a statically linked version of Busybox which is available for `mipseb`, `mipsel` and `ARM` as well as for many other architectures. We copy this Busybox binary into the guest filesystem and use it to set up a bridge, attach one network interface to it, configure both appropriately, and set up a default route (see Section III-B5). This approach yields considerably better results than relying on the default configuration of the guest system itself, as was done, for example, by Firmadyne.

D. Emulation

After the extraction and preparation of the filesystem, honware invokes QEMU to start the emulation. The honeypot is tested to see if it responds to ICMP echo request packets over the local network. If this is successful the necessary interfaces, routes and host firewall rules for connecting the honeypot to the Internet are created. We pre-route incoming packets on the host ethernet interface to the QEMU tap interface and post-route packets back to the host. By specifying which ports are handled this way, operators need only expose ports to the Internet that are of interest to them.

Honware can be configured so that the firmware emulation only runs for a certain period of time or *forever*, i.e. until stopped by the user. While honware is running it outputs kernel log information, details of incoming network connections, and all invoked commands with the relevant time information and writes this all to log files.

IV. EVALUATION

The evaluation of our framework is threefold. First, we compare honware with Firmadyne [11] in terms of extracted

firmware images, network reachability and number of emulated services. We do this by obtaining and running the exact same firmware images that they used. Second, we provide four case studies where we demonstrate that honware is capable of rapidly emulating devices to capture not only malware samples, but to emulate advanced device behaviour which is not feasible with traditional honeypots. Third, we perform extensive measurements to show that the performance of honware is comparable to real devices and that it is not susceptible to trivial fingerprinting based on timing attacks.

A. Extraction, network reachability and services

To measure how well honware extracts firmware images, configures the network and emulates services, we obtained the list of firmware images used in the evaluation of Firmadyne and downloaded all images that are still accessible on the URLs provided. As of March 2019, 8387 of 23035 images (36.4%) can still be downloaded. The list includes a variety of firmware images for CPE and IoT devices such as ADSL modems, routers, NAS systems, web cameras and smart power plugs. Unfortunately the authors of Firmadyne lost their database that would have allowed us to map the individual images to the outcomes *network reachable* and *listening services emulated* so we ran Firmadyne ourselves over the 8387 remaining firmware images to be able to compare results.

1) *Extraction*: As shown in Table I, honware appears better in extracting firmware images than Firmadyne. In total, we successfully extracted 4650 of 8387 available images (55.4%) compared to 2920 for Firmadyne (34.8%). We both use `binwalk` (the de-facto standard tool for firmware extraction) and for most manufacturers our results are very similar. Our significantly better results for Synology are probably because these firmware images are quite large, compressed to an average size of 133MB, and we allow filesystems up to 2GB in size (1GB for Firmadyne). That is, we suspect that Firmadyne failed to populate the filesystem correctly because of space constraints. The firmware images for which both Firmadyne and honware were unable to extract a filesystem are encrypted, have a proprietary way of packaging images or are simply updates – and so important folders and binaries are missing.

2) *Network reachability*: To measure network reachability, we prepared and ran all the successfully extracted firmware images as outlined in Section III-D and sent them ICMP echo request packets. As shown in Table I, we achieve significantly better results than Firmadyne. From the 4650 successfully extracted firmware images, 1903 images (40.9%) respond to the ICMP packets, compared to 460 images for Firmadyne (15.8% of the extracted images). For OpenWrt we were able to ping 674 devices whereas for Firmadyne no firmware image was reachable and this clearly increases our score. However, we find similar results for Zyxel (69 to 20), TP-Link (147 to 95) and Netgear (384 to 187).

We believe honware performs better for two reasons. First, Firmadyne supports only one ethernet device for their ARM little-endian platform whereas honware supports up to four.

¹`nmap -F -St -Su ipaddress`

TABLE I

COMPARISON BETWEEN HONWARE AND FIRMADYNE: WE OBTAINED THE LIST OF FIRMWARE IMAGES (23 035) USED IN THE EVALUATION OF FIRMADYNE (2016-02) AND DOWNLOADED ALL THAT REMAINED ACCESSIBLE (8 387) IN 2019-03. WE USED FIRMADYNE AND HONWARE TO EXTRACT THESE AND TEST THEIR NETWORK REACHABILITY BY SENDING THEM ICMP ECHO REQUEST PACKETS.

| # Brand | Available (2019-03/2016-02/ Δ) | | Extracted (honw./firm./ Δ) | | Network reach. (honw./firm./ Δ) | |
|-----------------|---|--------|---------------------------------------|-------|--|-------|
| 1 Actiontec | 0/14 | 14↓ | - | - | - | - |
| 2 Airlink101 | 0/15 | 15↓ | - | - | - | - |
| 3 Apple | 0/9 | 9↓ | - | - | - | - |
| 4 Asus | 1/3 | 2↓ | 1/1 | ← | 1/0 | 1↑ |
| 5 AT&T | 3/25 | 22↓ | 0/2 | 2↓ | - | - |
| 6 AVM | 0/132 | 132↓ | - | - | - | - |
| 7 Belkin | 123/140 | 17↓ | 49/49 | ← | 9/0 | 9↑ |
| 8 Buffalo | 97/143 | 46↓ | 6/7 | 1↓ | 2/1 | 1↑ |
| 9 CenturyLink | 13/31 | 18↓ | 7/4 | 3↑ | 7/0 | 7↑ |
| 10 Cerowrt | 0/14 | 14↓ | - | - | - | - |
| 11 Cisco | 0/61 | 61↓ | - | - | - | - |
| 12 D-Link | 1443/4688 | 3245↓ | 537/498 | 39↑ | 272/115 | 157↑ |
| 13 Forceware | 0/2 | 2↓ | - | - | - | - |
| 14 Foscam | 44/56 | 12↓ | 5/5 | ← | - | - |
| 15 Haxorware | 0/7 | 7↓ | - | - | - | - |
| 16 Huawei | 13/29 | 16↓ | 0/3 | 3↓ | - | - |
| 17 Inmarsat | 0/47 | 47↓ | - | - | - | - |
| 18 Iridium | 0/17 | 17↓ | - | - | - | - |
| 19 Linksys | 32/126 | 94↓ | 26/26 | ← | 15/1 | 14↑ |
| 20 MikroTik | 4/13 | 9↓ | - | - | - | - |
| 21 Netgear | 1396/5280 | 3884↓ | 639/629 | 10↑ | 384/187 | 197↑ |
| 22 On Networks | 0/28 | 28↓ | - | - | - | - |
| 23 Open Wir. | 0/1 | 1↓ | - | - | - | - |
| 24 OpenWrt | 756/1498 | 742↓ | 714/705 | 9↑ | 674/0 | 674↑ |
| 25 pfSense | 214/256 | 42↓ | - | - | - | - |
| 26 Polycom | 612/644 | 32↓ | 0/24 | 24↓ | - | - |
| 27 QNAP | 8/464 | 456↓ | - | - | - | - |
| 28 RouterTech | 0/12 | 12↓ | - | - | - | - |
| 29 Seiki | 0/16 | 16↓ | - | - | - | - |
| 30 Supermicro | 0/150 | 150↓ | - | - | - | - |
| 31 Synology | 1977/2094 | 117↓ | 1866/239 | 1627↑ | - | - |
| 32 Tenda | 6/244 | 238↓ | 4/3 | 1↑ | 2/0 | 2↑ |
| 33 Tenvis | 9/49 | 40↓ | 6/6 | ← | 6/4 | 2↑ |
| 34 Thuraya | 0/18 | 18↓ | - | - | - | - |
| 35 Tomato | 362/2942 | 2580↓ | 362/362 | ← | 217/0 | 217↑ |
| 36 TP-Link | 463/1072 | 609↓ | 171/171 | ← | 147/95 | 52↑ |
| 37 TRENDnet | 336/822 | 486↓ | 134/100 | 34↑ | 87/37 | 50↑ |
| 38 Ubiquiti | 26/51 | 25↓ | 20/19 | 1↑ | 11/0 | 11↑ |
| 39 u-blox | 0/16 | 16↓ | - | - | - | - |
| 40 Verizon | 0/37 | 37↓ | - | - | - | - |
| 41 Western Dig. | 0/1 | 1↓ | - | - | - | - |
| 42 ZyXEL | 449/1768 | 1319↓ | 103/67 | 36↑ | 69/20 | 49↑ |
| Total | 8387/23035 | 14648↓ | 4650/2920 | 1730↑ | 1903/460 | 1443↑ |

This is particularly important as a network bridge has to have at least one device attached (e.g. eth0) so that services are network reachable. Second, Firmadyne has no (fallback) mechanisms to correct missing network configuration settings, for example, because nvram could not be loaded in the absence of physical hardware. In contrast, as outlined in Section III-B5, honware will automatically detect an initial failure and will set up a bridge and an associated ethernet device.

3) *Services*: The execution of firmware applications is critical for honeypots since most exploits target these applications. If they do not function, then connecting firmware images to the Internet is of limited value. As shown in Table II, significantly

TABLE II

COMPARING HONWARE AND FIRMADYNE: TOP 15 LISTENING SERVICES.

| Prot. | Port/Service | Honware | Firmadyne | Δ |
|-------|----------------|---------|-----------|----------|
| TCP | 23/telnet | 879 | 149 | 730↑ |
| TCP | 80/http | 676 | 293 | 383↑ |
| UDP | 67/dhcp | 316 | 160 | 156↑ |
| UDP | 1900/UPnP | 239 | 128 | 111↑ |
| UDP | 53/various | 239 | 174 | 65↑ |
| TCP | 3333/dec-notes | 222 | 102 | 120↑ |
| TCP | 5555/freeciv | 203 | 57 | 146↑ |
| TCP | 5431/UPnP | 177 | 48 | 129↑ |
| UDP | 137/netbios | 154 | 82 | 72↑ |
| TCP | 53/domain | 139 | 73 | 66↑ |
| TCP | 443/https | 107 | 105 | 2↑ |
| UDP | 5353/mdns | 102 | 34 | 68↑ |
| UDP | 69/tftp | 104 | 26 | 78↑ |
| TCP | 1900/UPnP | 56 | 60 | 4↓ |
| TCP | 49152/UPnP | 53 | 62 | 9↓ |

more applications running under honware respond on their listening ports than it is the case for Firmadyne. We find that for telnet, 879 firmware images respond to our nmap scan compared to 149 for Firmadyne. HTTP on port 80 is the second most observed service with 676 firmware images responding to our nmap scan, followed by port 67 (316) and port 1900 (239).

We attribute our significantly better results to our instrumented kernel which has placed great emphasis on improved signal handling and on constructing a working network configuration, by executing our custom `/sbin/boot.sh` script. This technique allows us to very simply change default configurations, which is particularly important for firmware images that try to obtain IP addresses with DHCP.

B. Honeypot deployments in the wild

To evaluate the effectiveness of honware, we deployed multiple honeypots on the Internet including four brands of ADSL modems, TP-Link, D-Link, Eminent and ipTIME. We now discuss four case studies which show that devices can be rapidly emulated, very much faster than with previous approaches, and that honware can detect both known and previously unknown attacks. In particular, whilst emulating a router from ipTIME, we observed an unknown attack in which the default DNS setting in the router is changed to a rogue IP address – which we subsequently found to affect not only ipTIME, but also other brands.

1) *Broadcom UPnP Hunter*: UPnP Hunter is the name of a format string vulnerability in the Broadcom software for Universal Plug and Play (UPnP) and affects various brands, including TP-Link, D-Link and Netgear [3]. The vulnerability allows a remote adversary to cause the UPnP service to crash or execute arbitrary code. The attack seen in the wild was unusual in that an initial connection pre-qualified the devices as likely to be vulnerable before a second phase of the attack was attempted. 360Netlab reported that it took them over a month to code a custom honeypot to appropriately respond to each of the connections in turn [27]. With honware, because all services were operational, we were able to observe the

described attack within 24 hours of connecting the honeypot to the Internet.

To capture the attack, we emulated an ADSL router modem (D-Link DSL-2741B with firmware version 517b50, released on 2010-03-08). This router uses the MIPS architecture (big endian) and by default has listening applications on ports 21/tcp, 22/tcp, 23/tcp, 80/tcp, 1028/tcp, 67/udp, 69/udp, and 5431/tcp. Before we connected the emulation to the Internet, we used the proof of concept code [3] to test the exploit. The exploit uses the functions `SetConnectionType` and `GetConnectionTypeInfo` on port 5431, the UPnP SOAP service. The first function is used to set the format string and the latter one to read the output. As expected, we managed to cause the UPnP daemon to crash, read arbitrary memory and execute arbitrary code with root privileges, giving us full control of the device.

On 2019-23-01 a machine from India connected to our honeypot on port 5431 and sent `<NewConnectionType>.%08X.%08X.</NewConnectionType>` so as to exploit the vulnerability. We sent `<NewConnectionType>.7F8805AC.004332F0.</NewConnectionType>` to reveal the memory mapping. Subsequently a malware loader connected to our honeypot from the same IP address as observed by 360Netlab [27].

Unfortunately, the loader failed to download malware and instead sent a single character `X`. We are not sure why this happened: it may be that the attacker forgot to update a placeholder and include some malware or shell code, or that the attacker does not have shell code for the particular type of router that we used in our experiment.

Although we did not capture any malware, we have shown that we can, within a single day, create a honeypot for a particular device and observe an attack upon it. There is clear value in rapidly understanding complex attack vectors and shortening the time window in which attackers can abuse vulnerabilities without anyone being able to precisely identify their methods.

2) *DNS hijack*: We observed a previously unknown attack in which the default DNS server was changed within one of our honeypots that emulates an ipTIME N604R wireless router with firmware version 7.50 (released on 2011-01-31). This particular device is manufactured by EFM networks and is primarily used in Korea where it is distributed by ipTIME.

By default, the router has listening applications on port 80/tcp, 113/tcp, and 68/udp. In 2015, Kim [28] discovered a remote code exploitation vulnerability triggered by sending a crafted DHCP request. We were running two instances of this firmware on two different IP addresses located in Finland and Germany and were hoping to see this attack – instead of which we recorded a previously unknown attack.

The attacker used the `timepro.cgi` script and the WAN setup menu to overwrite the default DNS to a rogue IP address located in the Netherlands. This caused the device to change the iptables rule as follows: `/sbin/iptables -t nat -A PREROUTING -i br0 -d 192.168.0.1 -p udp --dport 53 -j DNAT --to-destination`

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:AddPortMapping xmlns:u="urn:schemas-upnp-org:service:WANIPConnection:1">
      <NewRemoteHost</NewRemoteHost>
      <NewExternalPort>47359</NewExternalPort>
      <NewProtocol>TCP</NewProtocol>
      <NewInternalPort>135</NewInternalPort>
      <NewInternalClient>192.168.8.1</NewInternalClient>
      <NewEnabled>1</NewEnabled>
      <NewPortMappingDescription>galleta silenciosa</NewPortMappingDescription>
      <NewLeaseDuration>0</NewLeaseDuration>
    </u:AddPortMapping>
  </s:Body>
</s:Envelope>
```

Fig. 2. EternalSilence: Malicious port forwarding rule captured by honeypot

X.X.X.X with x.x.x.x being a DNS server controlled by the attacker. They also configured a second DNS server to ensure that all DNS traffic went to their machines.

We experimentally resolved `www.yahoo.com` on the attacker’s DNS server and found that it resolved to a machine in China on AS41718 (China Great Fire Wall Network Limited Company). The resolved IP address has an unusual (self-signed) certificate which makes it authoritative for a range of websites, many with a Korean connection, but also `www.paypal.com`, `www.yahoo.com`, `www.google.com.tw` and many more. According to Shodan, there are 39 other IP addresses with the same certificate spread across the world, including Hong Kong, Taiwan and United States/California.

A search of online support forums showed that the same DNS servers were associated with other attacks. Three users with TP-Link² and Anderson³ devices had noticed that their DNS settings had been changed. We reported our findings to a vetted community of security professionals and law enforcement so they can take appropriate action.

3) *UPnPProxy*: EternalSilence is a newly discovered family of UPnPProxy forwarding malware [29]. EternalSilence adds port forwarding rules to compromised devices to expose TCP ports 139 and 445 behind routers. Every rule is added with an identical description of “galleta silenciosa” and can therefore be easily fingerprinted. The rules are persistently stored in the router’s configuration so reboots will not clear them; victims have to explicitly delete the rogue entries.

Using our framework, we set up an ADSL modem router (Eminent EM4544 with firmware version 8.38, released 2013-05-30). This type of router uses the MIPS architecture (big endian) and by default has listening applications on port 280/tcp, 113/tcp, 68/udp and 5431/tcp. We set up the honeypot on 2019-01-04 and four days later on 2019-01-08 a machine from Bolivia issued a M-SEARCH request (M-SEARCH * HTTP/1.1) followed by a GET request (GET /etc/linuxigd/gatedesc.xml HTTP/1.0) to retrieve the device

²<https://community.tp-link.com/en/home/forum/topic/158073?page=1&t=2019> and <https://trzepak.pl/viewtopic.php?f=20&t=61263>

³<https://eforum.idg.se/topic/358185-firefox-660-64-bit-quantum-sakerhetsvarnar-for-youtube-och-sokmotorn-duckduckgo/>

details. The response includes various device details such as deviceType, manufacturer and modelnumber. Subsequently the attacker issues an AddPortMapping request as shown in Figure 2. This rule triggers the miniupnp daemon to redirect port 47359 to 192.168.8.1:135, an action which is appropriately logged as follows: miniupnpd[202]: redirecting port 47359 to 192.168.8.1:135 protocol TCP for: galleta silenciosa.

We successfully captured the alteration of the port forwarding rules, but along with Akamai [29] who originally identified this attack, we saw no further attempt to exploit the compromised device. Nonetheless, honware is providing a mechanism to easily capture any such attack traffic by pretending to be a vulnerable device. If the malicious firewall rules are exploited in future, we will be able to observe this behaviour instantly and report it appropriately.

4) *Mirai variants*: The Mirai source code is constantly evolving and recently a new variant called Yowai/Hakai was found. This variant exploits a vulnerability in the *invokeFunction* of ThinkPhp [2] and allows the execution of arbitrary code on the underlying server. ThinkPhp is a PHP framework widely used by a variety of networked devices, particularly those manufactured in China [30]. Once devices are infected, they do further scanning to find other vulnerable devices. One advantage of exploiting ThinkPhp is that it does not compete with the original Mirai malware as it targets the web server on port 80, not telnet on port 23. Off-the-shelf Mirai honeypots would not record such attacks as they look for Mirai traffic on the telnet ports 23 and 2323 [31].

To capture attack traffic, we processed the firmware for the ADSL modem router TP-Link TD-W8960N, released on 2011-11-08, with honware and set up a honeypot. By default, this device has listening ports on 21/tcp, 22/tcp, 23/tcp, 80/tcp, 67/udp, 69/udp, 1900/udp and 5431/tcp. Our honeypot ran from 2019-02-17 to 2019-03-01 (14 days) and captured 566 attacks that tried to exploit the vulnerability in the ThinkPhp framework. In total, 49 different URLs, i.e. malware instances, were captured with a median of 4 attacks for each unique URL. One malware sample was associated with 70 of the attacks. We checked with Virustotal and 16 (32.7%) of the 49 samples were entirely new. Of the rest, over two thirds were captured by honware before they were first recorded by Virustotal; and only 13 (26.5%) samples had been detected by someone else and uploaded to Virustotal before honware. Across all 49 samples honware detected the malware a median of 9.7 days before Virustotal had a copy. It appears that we are able to make malware available to the defender community considerably faster than traditional honeypots.

C. Timing attacks to fingerprint honware

Fingerprinting is an ongoing concern for honeypot developers and operators. Once a honeypot is identified by attackers, its value in detecting new attack vectors and monitoring attack traffic will drastically decrease. It may also be that hosts running honeypots will be blacklisted by those attackers so that they become valueless, even for running undetectable

TABLE III
THINKPHP VULNERABILITY: TOP 15 MALWARE FILES OBSERVED WITHIN A 14 DAY PERIOD EMULATING A TP-LINK TD-W8960N

| #Seen | Filename | Country | First seen | | Detection ratio |
|-------|-------------|---------|------------|------------|-----------------|
| | | | Honware | Virustotal | Virustotal |
| 52 | Tsunami.x86 | DE | 2019-23-02 | unknown | 5/67 |
| 35 | cayo4 | DE | 2019-28-02 | 2019-21-03 | 10/68 |
| 34 | Tsunami.x86 | RO | 2019-19-02 | unknown | 5/67 |
| 8 | X86_64 | CA | 2019-28-02 | unknown | 0/66 |
| 6 | shiina | US | 2019-28-02 | unknown | 7/67 |
| 5 | Tsunami.x86 | US | 2019-27-02 | unknown | 0/66 |
| 5 | Tsunami.x86 | US | 2019-24-02 | unknown | 2/67 |
| 5 | lessie.x86 | NL | 2019-26-03 | 2019-23-02 | 2/66 |
| 4 | Tsunami.x86 | ZA | 2019-26-03 | 2019-01-03 | 13/71 |
| 4 | Tsunami.x86 | US | 2019-18-02 | unknown | 4/67 |
| 3 | Tsunami.x86 | DE | 2019-23-02 | unknown | 0/66 |
| 3 | Tsunami.x86 | US | 2019-21-02 | unknown | 2/66 |
| 2 | cayo4 | NL | 2019-22-02 | unknown | 0/66 |
| 2 | x86 | US | 2019-19-02 | unknown | 0/66 |
| 2 | Tsunami.x86 | US | 2019-27-02 | unknown | 1/66 |

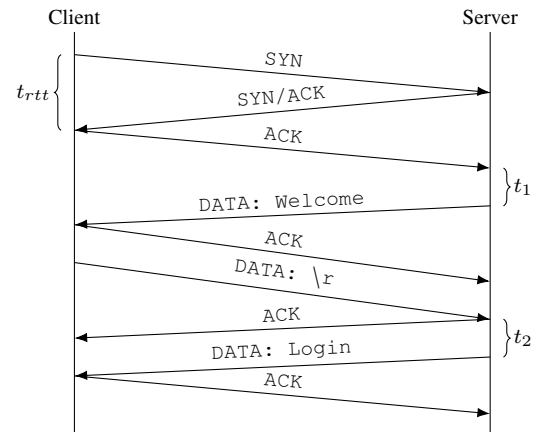


Fig. 3. FTP Server: We initiate a connection and measure t_1 and t_2 , the time the FTP daemon takes to respond to the ACK t_{ack} and the carriage return t_{cr} . Upon receiving the ACK, the remote server will send the welcome message 220 Welcome to ASUS RT-AC52U FTP service $t_{welcome}$ and the carriage return will cause the FTP server to present the login prompt t_{login} . In both cases, the RTT (t_{rtt}) is used to adjust the timing information on the received and transmitted messages so that $t_1 = t_{welcome} - t_{rtt}$ and $t_2 = t_{cr_{ack}} - t_{rtt}/2$ or $t_2 = t_{login} - t_{cr} - t_{rtt}$

honeypots. Thus honeypots should be always be built in such a way that they cannot be easily detected.

Honware is, by design, difficult to fingerprint at the network stack or application layer, but timing attacks are a potential concern – the emulation may significantly affect the speed of operation, so we evaluated this issue extremely carefully.

Emulation inevitably introduces overhead in terms of CPU usage, network latency and I/O operations. However, many CPE and IoT devices have limited resources; for example, the D-Link home router DIR 825 has a CPU clocked at 680 MHz and just 64MB of RAM. Furthermore, it is typically used in residential networks with limited (upload) bandwidth. In contrast, even the lowest tier virtual machines offered by popular cloud providers where honeypots might be deployed have a *virtual* CPU core clocked at several GHz, 1GB of RAM

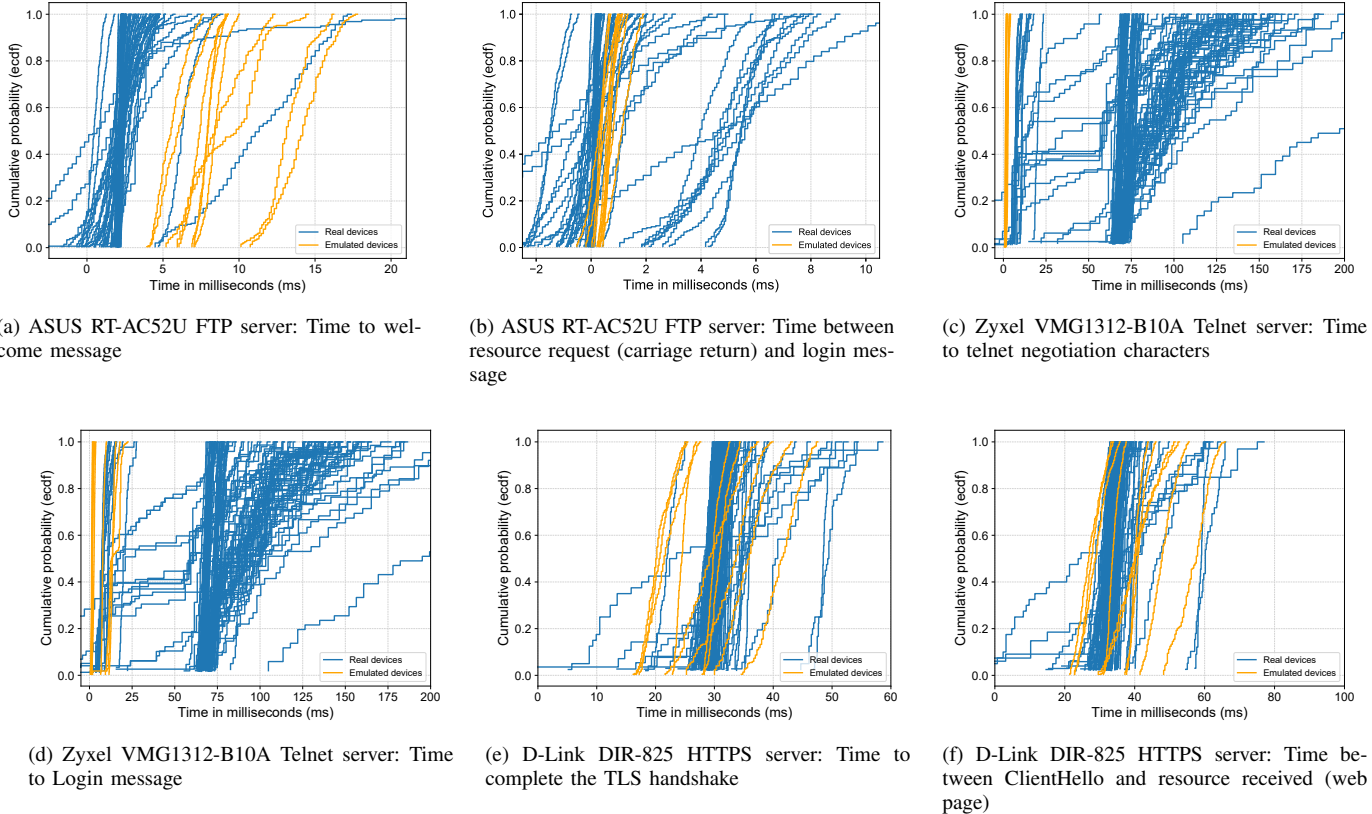


Fig. 4. Timing attacks against honware: We used three devices and protocols to measure the overhead of emulating honeypots with honware. Each line represents the empirical cumulative distribution function (ecdf) for one device. For each device, 300 measurements were made over a one day period to measure the time the application servers need to respond to our requests. To do so, the timing information is adjusted to factor in Round Trip Time(s) (RTT).

and a 1Gbit connection. Hence, any timing issue is as likely to result from running too fast as from running too slow.

To compare our honeypots to real devices, we sought out self-identifying devices with listening services on port 21, 23 and 443. We specifically chose 443 as encrypting traffic needs more resources and thus may serve as a good distinguisher between our honeypot and the real servers.

Unfortunately, most devices do not reveal their model and firmware version through their listening services without further interaction, but using Shodan we were able to find three suitable devices: the ASUS RT-AC52U Dual-Band AC750 wireless router (FTP server on port 21), the Zyxel VMG1312-B Wireless N VDSL2 Gateway (telnet server on port 23) and the D-Link Wireless N Dual-Band Router DIR-825 (web server on port 443). We assume that every device that returns the string VMG1312-B10A Login: when connected to on port 23 is the particular model in question, and Shodan reports receiving that string from 120 devices. Likewise, we expect the string 220 Welcome to ASUS RT-AC52U FTP service to be emitted only by ASUS RT-AC52U devices (74 devices) and the string HTTP/1.1 200 Ok Server: DIR-825 web server/v1.00 only to be sent by D-Link DIR-825 models (127 devices).

Having identified three suitable devices, we set up 30

honeypots to emulate them, ten for each, on two popular cloud providers with instances around the world including Singapore, Canada, USA, Germany, India, Netherlands and the United Kingdom. Then for each protocol (FTP, Telnet, HTTPS), we measure the time the application servers take to respond to our requests both for the honeypots and for the real devices identified via Shodan.

For each measurement, the initial round trip time (RTT), the time between the SYN and SYN-ACK packet, is calculated and is subsequently used to adjust the timing information on received and transmitted messages. As an example, figure 3 shows the interaction with the devices' FTP servers and the adjustments we made. Hence our measurements do not aim to identify network delays or Internet-induced latency, but solely to measure the time the application servers need to generate the appropriate response to an incoming probe.

When we connect to the FTP server on port 21, the remote server will send, upon completing the TCP handshake, the welcome message 220 Welcome to ASUS RT-AC52U FTP service at time $t_{welcome}$. Similar, after sending a further carriage return (t_{cr}), the FTP server will respond with 530 Please login with USER and PASS (t_{login}). The time it takes the application server to respond is therefore $t_1 = t_{welcome} - t_{rtt}$ and $t_2 = t_{crack} - t_{rtt}/2$

or $t_2 = t_{login} - t_{cr} - t_{rtt}$. The telnet and HTTP protocol are essentially similar, but instead of sending a carriage return, we start and complete the TLS handshake and subsequently ask the web server to send the main web page with a standard GET request. For telnet servers, we do not negotiate with the remote server or send any data other than the SYN and ACK packets as by default, telnet servers will start the negotiation process and present a login prompt without further interaction.

As shown in Figure 4, the application servers' response times do vary between the real and emulated devices. Each line in Figure 4 represents the empirical cumulative distribution function of one device in various locations as described above. For each of these devices, we made 300 measurements over a one-day period to measure the response time to our connections and resource requests. We find real systems that are faster than our emulated devices and systems that are slower – but with significant differences between protocols.

For FTP, the welcome message is consistently sent faster on real devices than on emulated ones. The latter need about 5ms to populate the welcome messages while the real systems took about 1.8ms. Interestingly, the message in response to our carriage return is not significantly slower on the emulated devices. We cannot be sure why this is the case, but we speculate that the ASUS FTP server performs additional operations for each initial FTP connection such as filesystem checks or initialising memory. These operations are likely to be particularly fast on real devices as they are using flash memory whereas the VMs use SSDs or even slower HDDs.

For telnet, the emulated devices respond very much faster than real ones. We further find that the response time for the real devices is fairly variable with most telnet servers responding in about 60-75ms (adjusted for RTTs) whereas emulated devices consistently took a few milliseconds. Similarly, the real devices need longer than the emulated devices to present the login prompt.

For HTTPS we find the real devices and emulated devices respond in about the same time with most servers completing the TLS handshake in about 30ms. The time between the start of the TLS handshake and the end of the application data transfer (web page) is also comparable.

Overall, we find that emulation does not generally slow down application servers – which we attribute to even the low-cost cloud instances we used having a far better specification than most CPE and IoT devices. Where emulation is faster, it would be possible to artificially slow honware responses.

It is of course true (and entirely expected) that in some instances it would be possible to fingerprint honware. However, the differences can be made small enough that it would take repeated measurements and a reference group (i.e. access to real devices) before an attacker could reliably distinguish an emulation from a real device. Furthermore, the Internet inherently introduces jitter, network delays and artefacts which all serve to further increase the time and effort to mount such attacks – and in the case of HTTPS, where honware is running at almost the same speed as the real devices – fingerprinting is going to be extremely problematic.

V. ETHICAL CONSIDERATIONS

We followed our institution's ethics policy at all times with appropriate authorisation at every stage. We reported the DNS hijack attack (Section IV-B2) to a vetted community of security professionals and law enforcement so they can take appropriate action.

Extracting firmware images to analyse their security properties locally, is long established practice [32], [33], [34], [35]. It may be argued that honware is different because we connect the firmware together with our custom kernel to the Internet where it is not then analysed by us, but by unknown malicious entities. Our view is that our research is in the public interest since being able to create honeypots rapidly for a variety of Internet-connected devices enables not only security researchers, but also manufacturers, to detect novel attack vectors and provide updates to patch vulnerabilities. Furthermore, we only use firmware images that are publicly accessible on the companies' websites and do not require registration or license keys.

We avoided doing our own Internet-wide scans to identify suitable devices for our timing measurements, but used Shodan instead. Our timing probe's contribution to the remote servers' overall traffic is negligible as we only initiated 300 connections for each device for a one day period. Devices that have open ports on 21, 23 and 443 will typically receive orders of magnitudes more traffic, in particular from malware (e.g., Mirai on port 23) or from search engines that index the web continuously.

When running high-interaction honeypots there is always the potential for damage to third parties, i.e. someone might use our emulated devices to conduct further attacks or perform malicious activities. This is a matter of significant concern, so we closely monitored our honeypots and stopped abuse once it became substantial in volume or of no further interest. For example, using our devices as a proxy to send spam emails is expected, but after the modus operandi, i.e. the attack vector, the attack itself and the consequences are well understood, we blocked all outgoing traffic and reset the device. For one device we had to block outgoing traffic as it was roped into a DDoS attack using the SSDP protocol. Our approach is completely in line with traditional good practices when running high-interaction honeypots.

It should be noted that taking actions to block outgoing traffic is a trade-off between understanding the attack scenario better and hiding from attackers who want to identify honeypots. An extended discussion about these trade-offs can be found in Nawrocki et al. [36].

VI. DISCUSSION

Honware is intended to identify attacks that cannot easily be captured with the traditional approach of low-/medium-/high- interaction honeypots. It is not designed as a tool for understanding large-scale, repetitive attacks, i.e. if a device is capable of being compromised by Mirai we are only interested in the attack once, not in seeing that the same attack occurs again and again every few minutes. After the attack vector is

known, we need to prevent further compromises, for example by blocking certain IPs or by recognising attack traffic (for example, Mirai’s initial scanning sets the Initial Sequence Number to match the destination IPv4 address). Once an attack is well-understood, medium-interaction honeypots should be set up to collect more quantitative data about large-scale attacks, reducing maintenance and minimising potential harm.

Honware’s real value is in its potential to rebalance the economics of attackers and defenders. It has become feasible to scan all of IPv4 address space for vulnerable devices with modest investment. Once an exploit is found for one technology, device, or specific implementation, attackers can easily find devices with that vulnerability – and instantly benefit from that exploit. Using honware to identify the exact attack vector and obtain copies of malware means that countermeasures can be deployed faster and with far more precision.

We accept that honware can be fingerprinted by attackers who are prepared to perform a significant amount of measurement work to identify small timing discrepancies. However, the focus of the work described in this paper is not to advance state-of-the-art sandbox anti-evasion techniques, but to develop a tool that enables the rapid construction of honeypots for a very wide range of devices – and for those honeypots not to be susceptible to fingerprinting attacks based on protocol deviations [19] or self-revealing properties [18]. Our approach of exposing the *real* services to the Internet and our use of the standard configuration files that are shipped by the manufacturers means that our honeypots will be indistinguishable from real devices. Traditionally honeypots were engaged in ‘Red Queen’s Race’ in which new fingerprinting attacks were countered by updating the emulation code. Honware avoids this entirely, which is particularly important as it has been recently shown that honeypot operators rarely update their honeypots or pay attention to how they are configured [37].

We envision honware being used at Internet scale, for example, by manufacturers setting up honeypots for every one of their products and firmware versions. They will learn whether known vulnerabilities are being actually exploited and they will learn of previously unknown issues in an extremely timely manner. Currently the number of potentially vulnerable devices listed on search engines such as Shodan is often used to classify vulnerabilities as low-, medium- or high- impact. The framework will allow a much better assessment of the risk of having (unpatched) devices connected to the Internet and allows for a more thorough approach in determining the impact of vulnerabilities.

At present the honware framework focuses on CPE and IoT devices but aims to support a wide variety of these devices. Thus we made certain design decisions which may not be optimal for a specific brand or device type. However, adjustments can be made at any point, in particular a cooperating device manufacturer could assist in specifying missing nvram values or by suggesting other configuration tweaks. Currently honware is limited to Linux-based devices for ARM and MIPS architectures. As other architectures become more prevalent, it is straightforward to recompile the Linux kernel. However, the

emulation is still limited by the capabilities of QEMU and its support for architectures and considerably more work would be needed to support devices which are not based on Linux, but use proprietary operating systems.

VII. RELATED WORK

IoTPOT was one of the first generic high-interaction honeypots tailored to impersonate IoT devices [12]. It supports eight architectures including ARM, MIPS and x86 and aims to return appropriate strings to connections on port 23. When the command is unknown, it tries to run the command in a sandboxed environment based on OpenWRT and infers the appropriate return string(s). Similarly, *Conpot* emulates industrial control systems based on the protocols Modbox and SNMP. It supports the emulation of large infrastructures so that adversaries may believe they are interacting with a complex industrial system network.

As IoTPOT and Conpot [38] do not use actual firmware to emulate devices and therefore return static information, Litchfield et al. developed HoneyPhy [39]. This framework tries to provide an appropriate simulation by taking a data feed from attached physical devices. For example, if an attacker turns on the heating via a compromised web-interface, the honeypot has to genuinely reflect these changes so that adversaries do not become aware that they are interacting with a honeypot. However, IoTPOT, Conpot and HoneyPhy only emulate specific application/ network layers and are not based on actual firmware. Thus their behaviour is bound to differ from actual IoT devices.

The approach of IoT CandyJar [40] is more sophisticated. IoT CandyJar utilises publicly available IoT devices on the Internet to collect responses for HTTP and then uses a Markov decision process (MDP) to respond to attackers’ probes. They show that, after a learning period scanning the Internet, they are able to send meaningful responses and capture attack traffic. However, their technique only works for non-encrypted traffic and can only capture responses before any login, i.e. router admin interfaces and similar cannot be represented. They also rely on finding a significant number of publicly available devices, which must also identify themselves, to provide meaningful responses.

In 2017 Guarnizo et al. [13] presented a “scalable high-interaction” honeypot platform based on physical devices. They exposed six security cameras, one networked video recorder and one networked printer through a distributed architecture on a range of IPv4 addresses. In the two months of their study they measured between 50 000 and 600 000 attacks on their devices, depending on location.

In 2016, vendors and ISPs were caught off-guard by the *TR-069 NewNTPServer* exploit which can be used to execute arbitrary commands on vulnerable routers [41]. TR-069 is an application layer protocol for remote management of end-user devices and custom honeypots that monitor TR-069 protocol are now available [42]. Similarly, new designs for programmable logic controller honeypots focusing on industrial control systems have been presented [43].

Recent advances have also been made by scanning the Internet IPv4 address space for vulnerable industrial control systems and identifying honeypots. Feng et al. [44] use a heuristic algorithm to determine the probability that the detected ICS device is a honeypot. More recently, it has been shown that industrial control systems are increasingly deployed around the world and that 60,000 thousand of these systems are publicly accessible [45].

Demonstrating the risk of IoT devices, Ronen et al. [46] showed that Philips Hue smart lamps can be used to spread malware. In their example, the malware spreads from one lamp to its neighbours and infects lamps located in the near vicinity. They estimate that for a city the size of Paris, only 15 000 randomly located light bulbs are sufficient to get every light bulb infected.

Closest to the present work Chen et al. [11] presented Firmadyne which dynamically analyses Linux-based firmware images to find vulnerabilities. They rely on a precompiled Linux kernel and use *QEMU* [47] as a full system emulator. However, their approach is not scalable as it requires constant manual effort and experts to classify failures during the firmware extraction and emulation phases. It is intended to allow developers to find vulnerabilities rather than to be deployed on the Internet to be probed by attackers. Our honeypot framework is significantly better in configuring the networking aspects of firmware images, in making the devices network reachable, and most importantly, in running the listening applications such as web servers (see Section IV-A).

Another project to focus on firmware images, *Firmalice* is a binary analysis framework that aims to find authentication bypass vulnerabilities [33]. It supports inspecting the codebase to find hardcoded credentials, hidden authentication interfaces and unintended bugs which allow adversaries to skip the authentication process and perform privileged operations.

VIII. CONCLUSION

Honware is the first system that allows system designers, developers and security researchers to efficiently and effectively deploy high verisimilitude, high interaction, honeypots for networked devices. Instead of having to buy and set up physical devices as honeypots, the framework facilitates the virtualisation of CPE and IoT devices merely by downloading standard firmware images from manufacturers' websites.

We demonstrate that honware can emulate a large variety of devices of many different brands within a virtual environment, independent of the underlying hardware. Our framework outperforms existing emulation strategies which are limited in their scalability, and honware is significantly better than previous projects in providing network functionality and in emulating the firmware applications – a crucial aspect as vulnerabilities are frequently exploited by attackers in 'frontend' functionality such as web interfaces or UPnP daemons.

An increasing number of exploits use multiple protocols in different phases of the attack and are targeted at very specific software implementations and devices. Generic honeypots are ineffective in capturing these attacks as they do not return

the appropriate traffic to allow later parts of the attack to commence and so be recorded. Honware uses the original firmware applications and their configurations which means that every phase of an attack can be monitored and fully understood.

Furthermore, using the original applications makes honware instances more fingerprint resistant and prevents fingerprinting attacks based on protocol deviations or those that identify configurations specific to honeypots. We further showed that the performance of honware is comparable to that of real devices and that it is not susceptible to trivial fingerprinting based on timing attacks.

Our honeypot framework has huge potential in detecting vulnerabilities in CPE and IoT devices that might otherwise be exploited for considerable periods of time without anyone noticing. We presented four real world case studies showing the practical value of our approach and in particular that within one day we were able to characterise a sophisticated attack which had taken experts a month to identify using traditional techniques. Additionally, while hoping to see an attack that had been reported to be occurring, we identified a previously unknown DNS changing attacker associated with a complex infrastructure.

Attackers are constantly scanning the Internet to find vulnerable devices. We believe honware is a major step forward in rebalancing the economics of attackers and defenders by cutting the attackers' ability to exploit vulnerabilities, particularly 'zero day' vulnerabilities, for considerable periods while defenders are unable to capture the details of the attack and thereby start the process of mitigation.

ACKNOWLEDGMENTS

This work was supported by the EPSRC [grant number EP/M020320/1]. We are grateful to Ross Anderson, Alastair R. Beresford, Alice Hutchings, Robert N. M. Watson, Daniel R. Thomas and Michael Dodson for helpful comments and discussions on the paper. We would like to thank Digital Ocean Inc for generous support in hosting our honeypot instances.

REFERENCES

- [1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis et al., "Understanding the mirai botnet," in *Proceedings of the 26th USENIX Security Symposium (USENIX '17)*. Vancouver, BC: USENIX Association, 2017, pp. 1093–1110.
- [2] TrendMicro, "ThinkPHP Vulnerability Abused by Botnets Hakai and Yowai," 2019. [Online]. Available: <https://blog.trendmicro.com/trendlabs-security-intelligence/thinkphp-vulnerability-abused-by-botnets-hakai-and-yowai/>
- [3] L. Juranic, "From Zero to ZeroDay Journey: Router Hacking (WRT54GL Linksys Case)," 2013. [Online]. Available: https://defensecode.com/whitepapers/From_Zero_To_ZeroDay_Network_Devices_Exploitation.txt
- [4] E. Bertino and N. Islam, "Botnets and Internet of Things Security," *Computer*, vol. 50, no. 2, pp. 76–79, 2017.
- [5] N. Provos, "A virtual honeypot framework," in *Proceedings of the 13th USENIX Security Symposium (USENIX '04)*, San Diego, CA, 2004, pp. 1–14.
- [6] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.

- [7] R. McGrew and R. B. Vaughn, "Experiences With HoneyPot Systems: Development, Deployment, and Analysis," in *Proceedings of the 39th Hawaii International Conference on System Sciences (HICSS '06)*, Kauai, HI, 2006, pp. 1–9.
- [8] S. Antonatos, I. Polakis, T. Petsas, and E. P. Markatos, "A systematic characterization of IM threats using honeypots," in *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS '10)*, San Diego, CA: Internet Society, 2010.
- [9] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *Proceedings of the 22nd USENIX Security Symposium (USENIX '13)*. Berkeley, CA: USENIX Association, 2013, pp. 605–619.
- [10] L. Spitzner, "The HoneyNet Project: Trapping the Hackers," *IEEE Security & Privacy Magazine*, vol. 1, no. 2, pp. 15–23, 3 2003.
- [11] D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards Fully Automated Dynamic Analysis for Embedded Firmware," in *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS '16)*. San Diego, CA: Internet Society, 2016, pp. 21–37.
- [12] Y. Minn, P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoT POT: Analysing the Rise of IoT Compromises," in *9th USENIX Workshop on Offensive Technologies (WOOT '15)*. Washington, D.C: USENIX Association, 2015, pp. 1–9.
- [13] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici, "SIPHON: Towards Scalable High-Interaction Physical Honeypots," in *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security (CPSS '17)*. Abu Dhabi, UAE: ACM, 2017, pp. 57–68.
- [14] Netgear, "Netgear Firmwares," 2017. [Online]. Available: <http://firmware.netgear-forum.com/index.php?act=interface>
- [15] TP-Link, "TP-Link Emulators," 2017. [Online]. Available: <http://www.tp-link.com/en/emulators.html>
- [16] Linksys, "User Interfaces," 2017. [Online]. Available: <http://ui.linksys.com/>
- [17] OpenWrt, "Linux Distribution for Embedded Devices," 2017. [Online]. Available: <https://openwrt.org/>
- [18] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Hernandez Ganan, M. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you... oh wait. An internet-wide view of self-revealing honeypots," in *Proceedings of the 17th IFIP/IEEE International Symposium on Integrated Network Management*. Washington DC, USA: IEEE, 2019, pp. 1–12.
- [19] A. Vetterl and R. Clayton, "Bitter Harvest: Systematically Fingerprinting Low- and Medium-interaction Honeypots at Internet Scale," in *12th USENIX Workshop on Offensive Technologies (WOOT '18)*. Baltimore, MD: USENIX Association, 2018, pp. 1–13.
- [20] T. Garfinkel, K. Adams, A. Warfield, and J. Franklin, "Compatibility Is Not Transparency: VMM Detection Myths and Realities," in *Proceedings of the 11th Workshop on HotTopics in Operating Systems (HotOS '07)*. San Diego, CA: ACM, 2007.
- [21] A. Kedrowsitsch, D. D. Yao, G. Wang, and K. Cameron, "A first look: Using linux containers for deceptive honeypots," in *Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense*, ser. SafeConfig '17. Dallas, TX: ACM, 2017, pp. 15–22.
- [22] T. Holz and F. Raynal, "Detecting Honeypots and Other Suspicious Environments," in *Proceedings of the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop (SMC '05)*, 2005, pp. 29–36.
- [23] S. Mukkamala, K. Yendrapalli, R. Basnet, M. K. Shankarapani, and A. H. Sung, "Network Based Detection of Virtual Environments and Low Interaction Honeypots," in *Information Assurance and Security Workshop (IAW '07)*, 2007, pp. 92–98.
- [24] Binwalk.org, "Binwalk," 2019. [Online]. Available: <https://github.com/ReFirmLabs/binwalk/>
- [25] L. Jaquemet, "SSLSNOOP," 2015. [Online]. Available: <https://github.com/trollldbois/sslsnoop>
- [26] /dev/ttyS0, "Emulating NVRAM in Qemu," 2012. [Online]. Available: <http://www.devttys0.com/2012/03/emulating-nvram-in-qemu/>
- [27] H. Wang and R. Kiter, "BCMPUPnP_Hunter: A 100k Botnet Turns Home Routers to Email Spammers," 2018. [Online]. Available: https://blog.netlab.360.com/bcmapupnp_hunter-a-100k-botnet-turns-home-routers-to-email-spammers-en/
- [28] P. Kim, "127 ipTIME router models vulnerable to an unauthenticated RCE by sending a crafted DHCP request," 2015. [Online]. Available: <https://pierrekim.github.io/blog/2015-07-06-127-iptime-router-models-unauthenticated-RCE-with-DHCP.html>
- [29] C. Seaman, "UPNPROXY: ETERNALSILENCE," 2018. [Online]. Available: <https://blogs.akamai.com/sitr/2018/11/upnproxy-eternalsilence.html>
- [30] ThinkPHP, "ThinkPHP5 Framework," 2019. [Online]. Available: <https://github.com/top-think/framework/>
- [31] M. Oosterhof, "Cowrie," 2019. [Online]. Available: <https://github.com/michelooosterhof/cowrie>
- [32] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "Avatar: A framework to support dynamic security analysis of embedded systems' firmwares," in *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS '14)*. San Diego, CA: Internet Society, 2014, pp. 1–16.
- [33] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalce – Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware," in *Proceedings of 22nd Network and Distributed System Security Symposium (NDSS '15)*. San Diego, CA: Internet Society, 2015, pp. 1–15.
- [34] A. Costin, A. Zarras, and A. Francillon, "Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces," in *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security (ASIACCS '16)*. Xi'an, China: ACM, 2016, pp. 437–448.
- [35] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "IoT Fuzzer: Discovering memory corruptions in IoT through app-based fuzzing," in *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS '18)*. San Diego, CA: Internet Society, 2018, pp. 1–15.
- [36] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A survey on honeypot software and data analysis," 2016. [Online]. Available: <https://arxiv.org/abs/1608.06249>
- [37] A. Vetterl, R. Clayton, and I. Walden, "Counting outdated honeypots: Legal and useful," in *Proceedings of the 4th International Workshop on Traffic Measurements for Cybersecurity (WTMC '19)*. San Francisco, CA: IEEE, 2019, pp. 224–229.
- [38] L. Rist, "Conpot," 2019. [Online]. Available: <https://github.com/mushorg/conpot/>
- [39] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos, and R. Beyah, "Rethinking the HoneyPot for Cyber-Physical Systems," *IEEE Internet Computing*, vol. 20, no. 5, pp. 9–17, 2016.
- [40] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang, "IoT CandyJar: Towards an intelligent-interaction honeypot for iot devices," in *Blackhat USA*. Las Vegas, NV: Blackhat, 2017, pp. 1–11.
- [41] J. B. Ullrich, "TR-069 NewNTPServer Exploits: What We Know so Far," 2016. [Online]. Available: <https://isc.sans.edu/forums/diary/TR069+NewNTPServer+Exploits+What+we+know+so+far/21763/>
- [42] Ö. Erdem, "HoneyThing – A honeypot for Internet of TR-069 things," 2016. [Online]. Available: <https://github.com/omererdem/honeything>
- [43] S. Lau, J. Klick, S. Arndt, and V. Roth, "POSTER: Towards Highly Interactive Honeypots for Industrial Control Systems," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. Vienna, Austria: ACM, 2016, pp. 1823–1825.
- [44] X. Feng, Q. Li, and H. Wang, "Characterizing Industrial Control System Devices on the Internet," in *Proceedings of the 24th IEEE International Conference on Network Protocols (ICNP '16)*. Singapore: IEEE, 2016, pp. 1–10.
- [45] A. Mirian, Z. Ma, D. Adrian, M. Tischer, T. Chuenchujit, T. Yardley, R. Berthier, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, "An Internet-Wide View of ICS Devices," in *Proceedings of the 14th IEEE Conference on Privacy, Security, and Trust (PST '16)*. Belfast, UK: IEEE, 2016, pp. 1–8.
- [46] E. Ronen, A. Shamir, A.-O. Weingarten, and C. OFlynn, "IoT goes nuclear: Creating a ZigBee chain reaction," in *Proceedings of the 38th IEEE Symposium on Security and Privacy (S&P '17)*. San Jose, CA: IEEE, 2017, pp. 195–212.
- [47] F. Bellard, "QEMU – A Fast and Portable Dynamic Translator," in *Proceedings of the 11th USENIX Annual Technical Conference (USENIX ATC '05)*. Anaheim, CA: USENIX Association, 2005, pp. 41–46.