HiFiPot: a High-Fidelity Emulation Framework for Internet of Things Honeypots

Pierre-Marie Junges, Jérôme François, Olivier Festor

HiFiPot: a High-Fidelity Emulation Framework for Internet of Things Honeypots

Pierre-Marie Junges, Jérome François, Olivier Festor
University of Lorraine, Inria, LORIA, France
Email: firstname.lastname@loria.fr

***Abstract*—Internet of Things (IoT) devices are easy targets for attackers. Preventing and counter-fighting this threat impose to capture and analyze attackers' behaviors. Several IoT-oriented honeypots have been proposed recently while, in parallel, emulation techniques for IoT devices have been improved to allow firmware analysis of this type of devices.**

In this paper, our objective is to consolidate these two facets in a single framework called HiFiPot. It is capable of creating a high-interaction honeypot on-the-fly with a high fidelity, *i.e.* from a firmware image. Our technique improves the most recent state-of-the-art solution to emulate an IoT device without scarifying the furtiveness. It is based on an iterative learning procedure to automatically correct emulation errors and to ensure Internet connectivity while maintaining these corrections invisible to the attackers. Out of the 1,000 firmware images tested, 443 (44,3%) can be deployed as honeypot. More than 500 instances of HiFiPot were deployed in the wild, and received about 1,900 HTTP traversal attacks, and downloaded 31 distinct malware binaries (out of 909) among which eight were unknown.

I. Introduction

The number of attacks targeting IoT devices keeps increasing, especially with large botnets [1]. To defend against an attack, understanding the attackers' behavior is the first step where honeypots play an important role [8].

Actually, several IoT honeypots have been proposed [4], [13], [16], [23], [27] with different architectures and objectives but generally assume that IoT devices include network devices such as gateways or routers [5]. In this paper, we thus refer to IoT devices or embedded devices without distinction and propose a new high-interaction honeypot system to provide the attackers with an environment as close as possible to a real one while the honeypot remains invisible as most as possible, like if the attacker was interacting with a real on-the-shelf device;

To provide a high-fidelity honeypot, our solution, HiFiPot, relies on QEMU [2] for emulating devices based on their firmware images. Because some vulnerabilities are specific to a device or a brand, a honeypot mimicking any off-the-shelf IoT device instead of a generic IoT honeypot running on OpenWrt [16] would lead to capture more specific attacks.

Potentially, any firmware image can be emulated, and then used as honeypot. Although such an approach is not dependent on the type of device (*e.g.* IoT), many issues arise when using QEMU without particular adaptation as already shown in [3], [10]. HiFiPot aims at improving the success rate of firmware emulation through an iterative learning process which refines an image until obtaining a bootable and stable image (adding missing files or mimicking NVRAM for example). However, this process has been carefully designed to avoid inflating the emulated device with useless and detectable files.

In addition to the improved emulation, HiFiPot also enhances common monitoring capabilities of honeypots, especially thanks to its capacity to decrypt and interpret on the fly SSL traffic, and its camouflage. Missing wireless network interfaces and files in `/dev` or `/proc` are also added dynamically.

The rest of this paper is organized as follows. Section II reviews related work. Section III starts with a high level description of the functionalities developed in HiFiPot. These components support the specific functionalities to enhance both the emulation capabilities (to build a correct QEMU image) and the honeypot-related functions (camouflage and attacker monitoring). The former and the latter are described in Section IV and section V respectively. Experimental results are presented in Section VI and compared to FirmAE [10], the most relevant, publicly available and recent IoT emulation framework. Ethical considerations are provided in Section VII before concluding in Section VIII.

II. Related Work

Over the years, many honeypots projects were developed for different purposes such as the analysis of telnet-based attacks [15]–[17], SSH-based attacks [15], [20] or Web-based attacks [13], [18], [21]. While primarily focused on attracting attackers looking for common services and machines, researchers have developed honeypots to deal with new threats, for example on mobile devices [22] or industrial control systems [12].

IoT honeypots are more difficult to develop because of the large variety of functionalities available in IoT devices (*e.g.,* telnet/ssh services, camera, web server). On one hand, a low-interaction honeypot must learn how to respond to a request. However, if a query is unknown then the attacker might leave the honeypot or worse, detect that the device is a honeypot. Therefore low-interactions honeypots for IoT have been focused on a limited set of services such as telnet [28] or UPnP [7]. To be more robust against attacker actions and to be less deterministic, adaptive strategies have been deployed for general purpose honeypots [24]. The concept has been extended to IoT devices in [27] where vulnerabilities are added

incrementally and credential modifications are applied. Alternatively, in [13], the authors propose to send unknown requests to IoT devices in Internet to learn the possible responses. However, the received responses might not be totally accurate because they can be from compromised IoT devices or other honeypots. Sending unknown requests to other devices may also spread attacks (*e.g.,* Remote File Inclusion attacks). As an alternative, the binaries present in the firmware images of IoT devices can be used to generate responses to attackers requests, notably to simulate web interfaces [26].

On the other hand, a high-interaction honeypot is a more sophisticated and complex kind of honeypot based on a real or emulated operating system with its functionalities. [14] presented a high-interaction honeypot specific to SSH only which intercepts the $< username, password >$ used during the SSH connection by modifying the OpenSSH[1] source code with loss of furtiveness. Instead, HiFiPot hooks the call to the original SSL library. The use of physical IoT devices connected to the Internet has also been explored to capture attacks [6], [19]. Nonetheless, this approach is not scalable and requires device-specific mechanisms to restore the compromised devices.

In [16], the authors developed a hybrid IoT honeypot consisting of a low-interaction frontend dealing with in-going telnet connections and a high-interaction backend composed of virtualized environments based on OpenWrt operating system. However, off-the-shelf IoT devices usually run vendor-specific Linux kernels and root filesystems leading their approach impractical to capture specific attacks.

We are interested in mimicking a device from its firmware image. In particular, extending QEMU emulator capabilities were investigated in [3], [10]. The authors in [23] follow a similar approach to build a honeypot and apply modifications to the firmware image to ensure a proper emulation. HiFiPot also uses an iterative learning procedure to derive a minimal number of modifications to make the honeypot highly similar to the original device but focuses on the mechanisms to hide the emulation part to the attackers. This requires to redesign the emulation process proposed in [3], [10].

III. Overview of HiFiPot

A. Functionalities

Our main objective is to build a honeypot representative of a real embedded device. Therefore, our solution relies on firmware emulation. On one hand, the emulation process takes as input an off-the-shelf firmware image that is then emulated using QEMU. On the other hand, a honeypot is supposed to act transparently in regards to the attacker while monitoring the latter. Hence, our framework is designed around two main objectives presented as follows:

- Enhance the emulation capabilities of the native QEMU emulator: even assuming only Linux based systems, there is a high variety in the supposed hardware or software and the firmware images highly differ from standard Linux distribution for common computers. As highlighted in previous approaches [3], [10], emulation may fail frequently. Hence, HiFiPot proposes a new set of improvements to construct a stable image such as adding missing files or configuring precisely the network interfaces.
- Enhance the honeypot experience by keeping its presence very stealthy while collecting relevant attacker traces and hiding the emulation framework.

[1]https://www.openssh.com/, last accessed on 10/19/2022

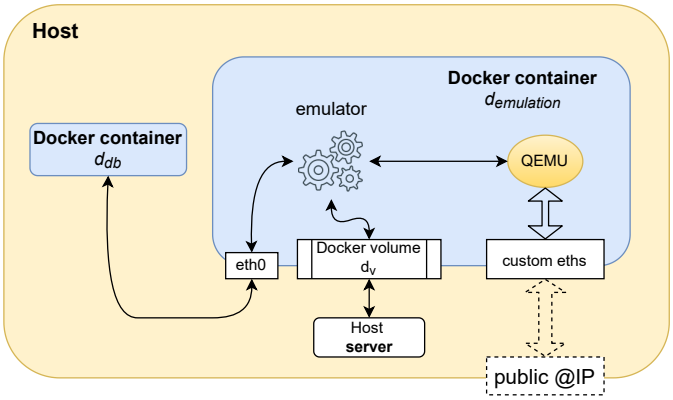


Fig. 1: HiFiPot framework

As highlighted, there is a strong coupling between these two sets of functionalities. On one hand, enhancing the emulator requires modifications to the original image which, in turn, reduces the stealthiness of the honeypot. Hence, some honeypot-related functionalities must compensate (shared library dissimulation and directory cover-up). On the other hand, adding missing files have to be done parsimoniously to avoid the honeypot to betray itself.

B. Architecture

As illustrated in Figure 1, our framework consists of a single host with two Docker containers: d_{db} is a database of firmware (firmware binaries, NVRAM values, device files' properties, etc.) and $d_{emulation}$ runs the QEMU instance, *i.e.* manages the emulation process. $d_{emulation}$ can be requested to create a honeypot from any image stored in d_{db}.

A Docker volume d_v is mounted onto an existing host's directory to allow synchronous (shared file descriptor of a UNIX socket) and asynchronous (shared files) communication between the host and the container. This directory contains the network traffic generated by a emulated device and the honeypot logs (attacker traces) transmitted through the network as introduced in Section III-D. The UNIX socket is leveraged to perform a coherent networking configuration between the container, $d_{emulation}$, and the host. This ensures the Internet connectivity of the honeypot (section IV-F).

We used a Linux kernel v2.6.36 compiled with a customized Linux Kernel Module (LKM) implemented to 1) improve the

emulation success rate and 2) enable the honeypot capabilities. This kernel version is one of most popular [25] in embedded device and is compiled for the 32-bits little-endian or big-endian MIPS architectures because they are the most common CPU architectures relevant to embedded devices [3], [23].

C. System call interception

Monitoring the attacker activities within a Linux-based Operating System (OS) requires the interception of, or to hook, the sensitive system calls (in kernel space). So, the `jprobe` and `kretprobe` debugging tools are used to intercept the desired function calls. The former is triggered before the function call with the same arguments whereas the latter is called once the function returns. This technique is inspired from FIRMADYNE [3] but HiFiPot hides the related kernel symbols and also removes the debugging files automatically created in `/sys` by these tools in order to maintain a high level of stealthiness and so avoiding the honeypot detection.

D. Attacker log transmission

Compared to [23], the observed attacker actions are not saved on the honeypot itself which makes HiFiPot stealthier. Instead, the attacker logs are transmitted over the network to the container using driver-specific network functions to bypass netfilter rules and so avoid the logs transmission being detected by attackers.

IV. Emulation Framework

A. Emulation process

Several steps have to be performed to build a stable QEMU image ready to be deployed from a packed firmware image, noted f_w. Firstly f_w is copied into the docker volume, d_v, between the docker container, $d_{emulation}$, and the host. Then, our emulation algorithm works as follows:

1) f_w is unpacked into uf_w using `binwalk`[2].
2) One subdirectory in uf_w having the greater number of UNIX directories (*e.g.,* `/bin`, `/usr`, `/etc` etc.) is selected as the root filesystem, noted $rootfs$.
3) A QEMU *raw* image, q, is created with a size equals to the two's nearest power compared to the size of $rootfs$. Then, the content of $rootfs$ is copied into q.
4) As inspired by [3], [10], one emulation-dedicated directory, noted d, containing for instance, our initialization script $init_{hifi}$, is created in q.
5) The iterative learning mechanism described in Section IV-B repeats n times the following actions; 1) emulates q, 2) analyzes the emulation logs looking for errors and warnings, and 3) updates q by, among others, configuring the network interfaces, adding missing files to solve these errors and warnings.
6) Finally q is emulated as a honeypot during a predefined period of time or until a user interruption.

[2]Available at https://github.com/ReFirmLabs/binwalk, v2.2.0-ff34b12

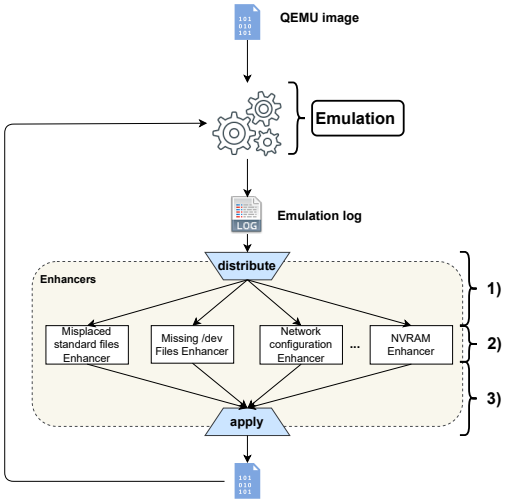


Fig. 2: Iterative learning from the emulation logs

Considering the importance of $init_{hifi}$ and lib_{shared}, they are placed into d, hidden from the attackers. This directory also contains an *auxiliary* script completed in step (5) to enable network connectivity and perform user-given functions. Thus, the *auxiliary* script is executed by our LKM upon request.

B. Iterative learning

To achieve our objectives, we created one Linux Kernel Module (LKM), noted m, able to intercepts up to 20 system calls using the techniques described in Section III-C. More precisely, two different kernels are built. The first one is configured to be highly verbose with many hooks to observe raised issues, errors or warnings during emulation logs that are then used by the iterative learning procedure. Once the refinement done, the honeypot deployment uses a second kernel having solely the hooks managing the honeypot functionalities.

Compared to the existing solutions [3], [10] creating all the commonly missing `/dev` and `/proc` files, our approach resides in building a minimal filesystem by identifying the required `/dev` and `/proc` files, the misplaced files or the modules expected by an embedded device. To do so, HiFiPot emulates iteratively the same firmware while updating the root filesystem according to the errors that occurred. As illustrated in Figure 2, it distributes error logs to multiple *Enhancers* to correct the emulation problems. An *Enhancer* is actually a `python` class defined to interpret emulation logs and update the device filesystem according to a particular type of errors. So, HiFiPot has been designed to be extended.

By default, HiFiPot provides the following *Enhancers* described in details in the next sections:

- The *Shared library compilation Enhancer* compiles our shared library with different functionalities depending on the device filesystem.
- The *NVRAM Enhancer* ensures the NVRAM mimicking.

- The *Misplaced standard files Enhancer* assures that missing or misplaced files are relocated according to the device expectation.
- The *Missing `/dev` files Enhancer* creates the device files requested by the device with customized `ioctl` return values if necessary.
- The *Missing `/proc` files Enhancer* manages to fake outputs for certain configuration `/proc` files and also creates the missing ones.
- The *Network configuration Enhancer (NCE)* ensures the connectivity of the emulated device with the Internet. This particular enhancer requires to update the configuration of the Docker container.

C. Shared library compilation

SSL on-the-fly decryption and HTTP access restriction are achieved within a shared library as they are executed in the user space. However, they must be enabled only if prerequisites described in Section V-B are met. Hence, the content of the `/lib`, `/usr/lib` and `/usr/local/lib` directories present in the device filesystem are analyzed by the *Shared library compilation Enhancer* to detect if the shared library must implement or not these features. Thus, the NVRAM interactions are also achieved using the shared library. Once the shared library is specified and compiled, this *Enhancer* requests the HiFiPot LKM to ensure that the shared library is 1) loaded through the `LD_PRELOAD` environment variable to every newly started process, and 2) hidden from the attacker.

D. Non-volatile Random Access Memory Mimicking

Real hardware devices contain non-volatile random-access memory (NVRAM) used to retrieve or store configuration values. Such values are not always included in the firmware. Like [3], HiFiPot uses an NVRAM mimicking mechanism based on the shared library lib_{shared} implementing the functions to store and load the NVRAM $< key, value >$ entries using a standard UNIX directory. Thanks to the request generated and sent to the LKM, as described in Section IV-C, lib_{shared} is able to intercept the NVRAM-related function calls to all the processes started in an emulated device.

Unlike [3], [10], the NVRAM entries are not hard-coded but stored into a file automatically added to the C code via macros. Hence, managing the NVRAM (changing entry values for example) is possible for non-expert users. HiFiPot contains a set of 26 default entries, such as the WAN or LAN network interfaces or the default IP address to use. Thus, our technique also detects attempts to access unknown entries which are, in turn, created dynamically by the *NVRAM Enhancer* with random or zeros data in order to avoid a program crash.

E. Missing files

In order to ensure an emulated device appears similar to a real one with high fidelity, `/dev` and `/proc` files must be carefully crafted since they are partially related to the embedded hardware (*e.g.,* driver files). Indeed, naively creating all the commonly `/dev` and `/proc` files may be used by attackers to detect the honeypot.

Therefore, HiFiPot keeps the honeypot stealthiness intact by adding the minimal set of necessary files. HiFiPot stores `/dev` or `/proc` files into four tables in the main database: (1) a table[3] storing the minor, major, type and name of standard `/dev` files; (2) a table storing the brand-specific data to create `/dev` files; (3) a table storing the brand-specific data for `/proc` files; (4) a table storing non-brand specific information for `/proc` files. For (2), (3) and (4) data must be populated manually or using other sources of information like [3], [10].

Based on this database, HiFiPot detects the missing files using a `jprobe` and a `kretprobe` on the `do_sys_open` system call. The former identifies the opened files whereas the latter is helpful to detect the `ENODEV` errors indicating that a file f is missing. Then, its filepath is written into the emulation log.

In particular, once the emulation terminates, three *Enhancers* use this information as follows

1) if the filepath starts with `/dev`, the *Missing `/dev` files Enhancer* searches f in (1) and (2) for a matching entry. If an entry is found in (1) (standard devices), one `mknod` command is generated and added to our initialization script $init_{hifi}$. Otherwise, the `/dev` file will be created according to the content in the matched entry in (2) using a request to the HiFiPot LKM also added to $init_{hifi}$.
2) If the filepath starts with `/proc`, the *Missing `/proc` files Enhancer* looks for f in (3) and (4) and extends $init_{hifi}$ to create the `/proc` file accordingly.
3) Otherwise, the *Misplaced standard files Enhancer* searches the whole firmware image for a file having the same name as f and copies its content into f if it exists.

It is worth mentioning that when creating a `/dev` or `/proc`, our LKM allocates a `file_operations` structure with the `read`, `write`, `open`, `ioctl` functions. The latter, only available for the `/dev` files, is often used by programs to interact with a given device file. Hence, when creating a `/dev` file using the HiFiPot LKM, the expected return values and conditionals to use in the `ioctl` function can be specified.

F. Network configuration

As stated in Section III-B, our emulator is located in the Docker container $d_{emulation}$. A UNIX socket interconnects the latter with the host h. This section focuses on enabling the network interfaces and connectivity of the QEMU emulated device ed. This is achieved thanks to two levels of interconnections: $ed \leftrightarrow d_{emulation}$ and $d_{emulation} \leftrightarrow h$. For each level, network configuration with valid interfaces, IP addresses and routes must be set.

To guarantee a high fidelity in regards to the emulated device, we excluded to simplify the emulated device configuration

[3]populated using the data available at https://www.kernel.org/doc/Documentation/admin-guide/devices.txt

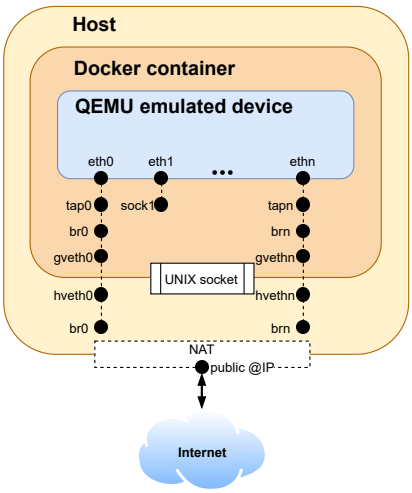


Fig. 3: Network interfaces

with a single preconfigured interface although this is sufficient for emulation purposes only [3], [10]. Hence, one or multiple network interfaces have to be configured according to the firmware including wireless interfaces.

1) Emulated device-container network configuration: To interconnect the emulated device and the container, HiFiPot first identifies the network entities to be configured at the device side by looking for network-based operations performed during the startup of the firmware (*e.g.,* creation of routes, bridges, VLANs). Hence, HiFiPot intercepts the call to the following functions when the firmware is running:

- ip_rt_ioctl, fib_magic } Add, delete or modify a route
- inet_bind: Gather information about the opened services
- register_vlan_dev, register_vlan_device } Assign VLAN id
- __inet_insert_ifa: Assign IP address
- br_add_if:, br_dev_ioctl: } Create and manage bridge

The arguments of these functions are transmitted via the log filtering operation described in Section IV-B to the *Network configuration Enhancer (NCE)* which constructs, as illustrated in Figure 4, a graph from the extracted information to create links between interfaces, bridges and routes (and optionally with VLAN) such as:

1) ethernet interface $\rightarrow$ ethernet interface.{VLAN id}
2) ethernet interface {,.VLAN id} $\rightarrow$ bridge
3) any interface $\rightarrow$ route

From the initial graph, our approach creates the missing edges and nodes such that all ethernet interfaces are connected to a valid route and adds a default route if none exists. Then, each path starting from an ethernet interface to a route node is then translated to BusyBox's `ip` and `brctl` commands which are then added to the *auxiliary* script.

Thanks to our approach, the number of ethernet interfaces is known and so, the connected network interfaces at the container side (*tapX* and *sockX* in Figure 3) are created.

2) Container-host network configuration: By default, in bridge mode, the Docker container is associated to one network namespace, noted *NS*, with one ethernet interface `eth0` which is simply a virtual ethernet device (veth) pair between the host network and *NS*. The latter is modified according to Figure 3. For *socket* interfaces with no connectivity, no further action is required. For the *tap* interfaces, we create a veth pair and bridge it to the interfaces on each side. This creates a tunnel when forwarding and IP masquerading is enabled. Because such modifications have to be made by the host, the *NCE* sends the instructions to the host using the UNIX socket.

Because our final purpose is to deploy honeypots, packets interacting with one predefined server's network interface are forwarded to the emulated device using netfilter pre and postrouting rules.

G. Schedule user-defined actions

A user may be willing to perform particular actions on the emulated device or update the device filesystem to change binaries such that, in honeypot-mode, attackers can be trapped in an improved environment. For example, as shown in [10], forcing the execution of `iptables` commands to clean firewall rules improves the emulation success rate. Hence, to keep our emulation framework brand-agnostic and highly flexible, the *Task Enhancer* relies on an additional JSON file completed by the user to execute, replace or modify any file present in the emulated device.

V. Honeypot functionalities

A. Monitoring the attacker activities

The network traffic generated by the honeypot is saved in PCAP files. In addition, different monitoring modules have been defined to associate the attackers' actions in the honeypot with the traffic they generate.

1) Monitoring the tty activities: Intercepting the inputs written by the attacker in a terminal or pseudo-terminal is helpful to understand the attacker's actions and behaviors. Rather than only keeping track of the inputted commands, we are also interested in distinguishing automated attacks, dummy attackers making copy-paste or expert attackers typing commands in live. Our aim is thus to capture every single key pressed associated rather than solely the full-typed commands.

The pressed keys are retrieved using the `n_tty_receive_buf` function provided by the `tty` driver because of its arguments that are directly accessible (*i.e.,* not in userspace). Thus, keyboard shortcuts sent over a terminal (*e.g.,* `ALT+F4` or `CTRL+C`) are also intercepted.

2) Capturing attacker downloaded files: Once the attacker accesses the honeypot, it is most likely to observe the download of additional software or scripts to exploit a vulnerability using tools like `wget`, `curl` or `tftp`. Attackers can also rename binaries such as `wget` to make its usage more stealthy [16]. Hence, only monitoring specific programs is unreliable.

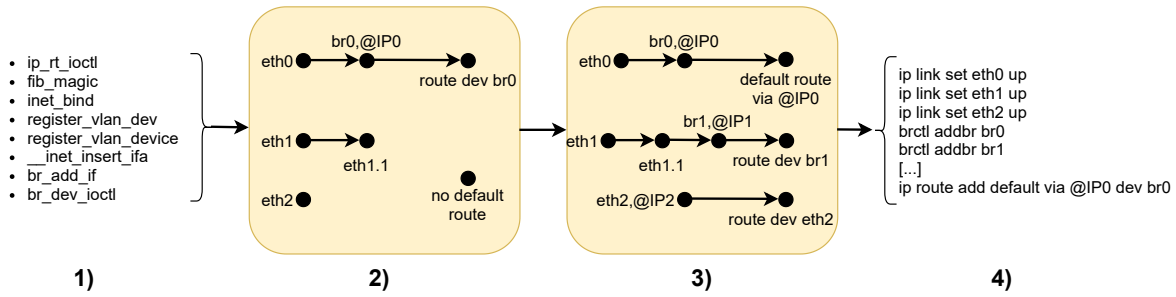


Fig. 4: *Network configuration Enhancer (NCE)* algorithm

It is preferable to intercept relevant information at a lower level. We assume that a program downloading a file (1) opens at least one connection and (2) writes its content to the disk. This behavior is identified using one `jprobe` on the `sys_write` system call. Finally, three `jprobes` on `sys_close`, `sys_exit` and `sys_signal` are used to detect whether or not the writing process ended.

B. On-the-fly SSL decryption and HTTP access restriction

Intercepting the attackers activities over encrypted channels is challenging. HTTPS decryption is usually performed offline using the certificate generated by the emulated device [23]. However, web interfaces are commonly accessed with HTTPS now and used to display and control device-specific elements (sensors, video stream, etc.). As a result, the web interface would be very incomplete or erroneous revealing the honeypot to the attacker. Hence HiFiPot restricts the access to the login page by transparently modifying the credentials received which requires on-the-fly decryption of the HTTPS traffic.

For that purpose, the HiFiPot shared library lib_{shared} includes a hook on the `SSL_read` function call as it is used by the applications to decrypt and read data. The decrypted data are then parsed and modified if known credentials have been found, otherwise the decrypted data are returned directly. Similarly, in case of plain text (*i.e.,* HTTP request), hooks on `recv`, `recvfrom` and `recvmsg` intercept the data and perform the same modifications described above.

C. Shared library dissimulation

Our shared library lib_{shared} is included via the `LD_PRELOAD` environment variable to every process. By checking the environment variables, an attacker would easily detect the emulation framework and so, the honeypot. To mitigate this issue, we implemented in our LKM two methods to remove the lib_{shared} information from 1) the environment variables (*e.g.,* `/proc/{self,pid}/environ`), and 2) the memory mapping of each process (*e.g.,* `/proc/{self,pid}/maps`). Thus, the latter method also ensures that the showed memory addresses remain contiguous.

D. Emulation directory cover

One directory, noted d, is created on the emulated device to store the elements necessary for the emulation as described in Section IV-A. However, in a honeypot context, leaving such a directory in plain sight would alert the attackers.

TABLE I: Dataset brand distribution

Brand	Occurences	Brand	Occurences
Netgear	279 (27.9%)	D-Link	129 (12.9%)
Ubiquiti	200 (20.0%)	Asus	96 (9.6%)
Tp-Link	131 (13.1%)	Linksys	18 (1.8%)
Trendnet	131 (13.1%)	Edimax	16 (1.6%)

Hence, our module modifies the `filldir_t` function given as an argument to `vfs_readdir` such that d is ignored when completing the list of directories present. Furthermore, the name of d is randomly generated at every deployment.

E. Dummy network interfaces

Embedded devices often have wireless network interfaces such as `wlan0` or `ra0`, which are not supported in the last QEMU stable version (7.1.0). However, the lack of such interfaces may be searched by the attackers to detect if they are in an emulated environment and not a real device.

HiFiPot automatically creates the missing wireless interfaces through the LKM. In a first step, the function `do_vfs_ioctl` is hooked to retrieve `ENODEV` (*i.e.,* no such device) failures when the emulated devices tries to setup the wireless interfaces. In a second step, Linux *dummy* interfaces are registered in the kernel space accordingly and affected with random statistics (number of bytes received for example).

VI. Evaluation

In this section, our objectives are to (1) assess the capability of HiFiPot to correctly emulate various firmware images and (2) to introduce a preliminary in-the-wild deployment.

A. Experimental setup

The authors of [9] provided us with their dataset of 4,730 firmware images released between 2009 and 2019 by major manufacturers. As shown in Table I, we randomly selected 1,000 firmware images having MIPS (little or big endian) as CPU architecture.

HiFiPot was deployed on two servers $S1$ and $S2$ running on a Ubuntu 18.04 LTS with a Xeon E5-2620v3 2.40GHz and 128GB of RAM, and 16.04 LTS with a Xeon E5-2420v2 2.20GHz and 64GB of RAM respectively. On average,

HiFiPot can rapidly deploy any new firmware image in about 5 minutes making the honeypot highly flexible. Note that the number of trials in the iterative learning procedure was bounded to 3 after preliminary testing on a few firmware.

Once the image is built, a device (*i.e.,* firmware image) is firstly emulated without forcing any processes to start (*native* mode). In case of failure, a first arbitration consists of a `sleep` process started in parallel to the *init* script to force the device to hang allowing other binaries to start. Finally, as inspired by [10], if the device remains unreachable, the second arbitration the *FirmAE's like* combines the *force-hang* with the execution of *iptables* commands to clean the firewall rules.

Obviously, the *native* mode is the most stealthy and most compatible with honeypot usage. However, the most advanced configuration is useful to compare HiFiPot to a baseline tool, FirmAE [10], that is more recent and has been proven more effective than FIRMADYNE [3]. To the best of our knowledge, the authors of honware [23] have not provided details about honeypot intrinsic camouflage and the source code is not publicly available. Hence, we cannot compare fairly our approach with this solution.

1) Reachability: Our first experimentation assesses the reachability of an emulated device d. This test checks that at least one of its interfaces is accessible from the Internet. 25 *ICMP echo request* probes with 0.5 second timeout are sent as a test 100 seconds after the device booted.

The ratio of reachable emulated devices is reported in Figure 5(a). The success rate of HiFiPot significantly increases from the *native* to the *force-hang* modes independently of the brand but notably in the case of Edimax with an increase from 6% to 94%. Indeed, considering that an emulated device fails at startup, it may have crashed with an *attempted to kill init* kernel panic after its *init* execution. By forcing the execution of a `sleep` process, the device remains alive.

On one hand, as used in [10], the *firmAE-like* mode which cleans the `iptables` rules in complement to the `sleep` process, improved significantly the capability to emulate Tp-Link devices by 63.63% indicating that Tp-Link usually set default firewall rules. On the other hand, our approach fails to properly emulate about half of the Ubiquiti and Asus devices. Thanks to a manual analysis, we concluded that the Ubiquiti's device failures are mostly due to a specific module that fails to be inserted and ASUS uses non UTF-8 NVRAM keys which are not correctly interpreted by HiFiPot when the verbose mode is enabled. Nonetheless, these two brands cannot be successfully emulated with FirmAE neither.

However, in other cases, the success rate of HiFiPot is between 84% and 96% (*firmAE-like*) outperforming FirmAE with a limited success rate between 60% and 67%. HiFiPot is thus very adapted to emulate various firmware.

2) Active services: Using `nmap`[4], we counted the number of open ports among the first 1024 TCP ports of the emulated device. This measures the correct execution of the function calls run by the emulated devices. If the emulated device runs a service, this implies that the corresponding function call called during the device's initialization met all its requirement.

TABLE II: Top 10 targeted services

Service	Port	Protocol	Nb. Packets
telnet	23	tcp	4,951,185 (62.82%)
SSH	22	tcp	1,033,778 (13.12%)
HTTP	80	tcp	85,694 (1.09%)
HTTPS	443	tcp	8,820 (0.11%)
NetBIOS	445	tcp	7,162 (0.09%)
DNS	53	udp	6,311 (0.08%)
HTTP	81	tcp	2,504 (0.03%)
NTP	123	udp	1,457 (0.02%)
LDAP	389	udp	817 (0.01%)
NetBIOS	139	tcp	790 (0.01%)

Figure 5(b) shows the number of devices running at least one service. Due to the same reason explained in Section VI-A1, most of the Ubiquiti devices failed to run at least one service showing that this brand would require specific emulation adjustments. Overall, we observed that for all arbitrations about 40% of the reachable devices run at least one service compared to FirmAE having 83.92%. However, FirmAE, with full arbitrations, also force to start an HTTP server if it exists in the device. Despite this bias in favor of FirmAE, HiFiPot is also able to emulate a device with more active services for certain cases without forcing their executions (*i.e.,* runs by the emulated firmware itself) as highlighted in Figure 5(c).

As described in Section IV-F1, the network interfaces of the honeypot are configured by the *auxiliary script* which is executed 60 seconds after the start-up, so certain binaries (*i.e.,* services) may fail to start due to the lack of configured network interfaces. Therefore our more generic approach should be reinforced by user-given commands as presented in Section IV-G, for example to also force a web server to start.

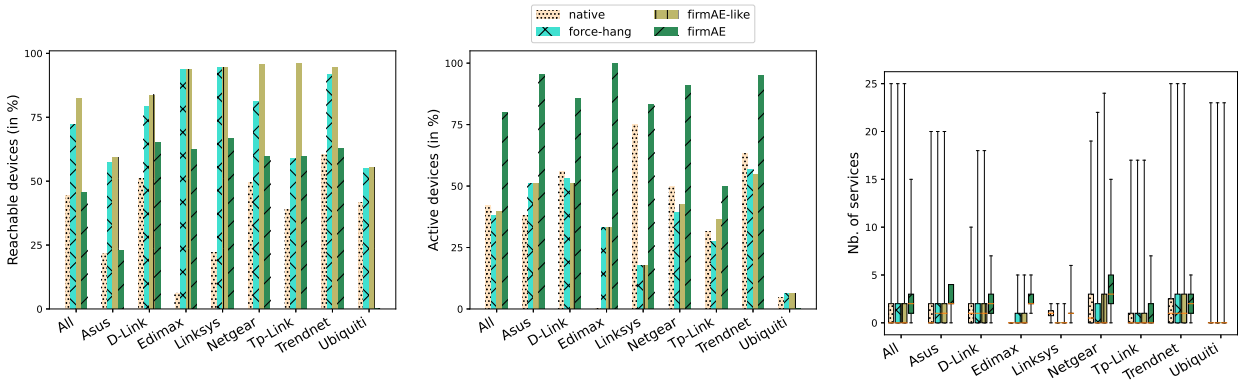
B. Honeypots in-the-wild

This last experiment is a preliminary evaluation of deploying hundreds of different honeypots using HiFiPot. The objective here is to assess if the honeypot related functionalities are helpful to track attacker behaviors. We expect the deployed honeypots to have a high stealthiness and so be capable of capturing real (human) attackers and new malware in addition to massive and common attacks and binaries such as the Mirai botnet [1], [11]. We rely on the keys pressed such as `backspace` or control keys to detect a human as explained in Section V-A1 and on VirusTotal[5] (VT) to identify if new malware has been downloaded. To detect HTTP traversal attacks, regular expressions are used on the HTTP payload to find patterns such as `../../*` or `*/proc*`.

In a first experiment, 500 firmwares were randomly selected deployed for 3 hours each between May and August 2021. We have collected 193MB of PCAP files with 25,308 unique IP addresses trying to connect to the honeypots. A large majority

[4]https://nmap.org/

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



(a) Number of reachable device (*reachability*) (in %)

(b) Number of device with at least one active service (in %)

(c) Average number of active services

Fig. 5: Emulation results

of those were only scanning our devices. Actually, only 2,643 (10.44%) IP addresses have sent more than ten packets. For 6 of them, we have found evidence of human behavior. Even though those numbers seem low, it is worth mentioning that the deployment of each honeypot was rather short (3 hours) and the firmware selection was done randomly. Thus, the selected firmwares (*i.e.,* devices) may not expose a service providing remote access, like telnet or SSH.

Therefore, in a second experiment, 138 firmware images with the services telnet, SSH or HTTP(S) were repetitively deployed for 3 hours each between September 2021 and January 2022 in a round robin manner. More connections have been observed which resulted in observing 144,272 unique IP addresses among which 9,531 (6.6%) sent more than 10 packets to the honeypot. As highlighted in Table II, telnet and SSH represents 75% of targeted services. This confirms that these services, known to be easily compromised, are still very exploited by attackers. Deploying honeypot with these services active (as we did in this second experiment) is thus needed in order to track major attacks.

In our case, 45 IP addresses are considered as potential human attackers. Notably, we observed an attacker trying to use the *escape* key or exotic control commands to exit the login interface of the telnet service. Although we did not observed any attacker inspecting precisely neither the device hardware nor software components, our honeypot was not detected by Shodan's honeyscore[6]. 31 distinct malware binaries (out of 909) were downloaded among which eight were unknown and detected as malicious by VT whereas 15 others were uploaded less than a week after their first upload on VT. Our honeypot is thus capable of fresh large-scale attacks.

Regarding the HTTP attacks, we observed 85,694 HTTP connections from 4,859 distinct IP addresses among which 51 attempted 1,885 HTTP traversal attacks. Actually, 834 unique HTTP payloads have been observed whose the main objective was to recover the passwords from the honeypot (6 out of top 10 unique payloads).

As a conclusion, HiFiPot allows the deployment of high interaction honeypots which are both capable of trapping real human attackers and attracting new and recent malwares.

VII. Ethical considerations

To preserve the high fidelity likeness with the original device, the emulated environment allows an attacker to perform all types of operations like any other high-interaction honeypot. The honeypot can thus serve to scan other hosts, participate in a DDoS or spread malware. Our honeypot infrastructure can represent a threat for outside hosts. This risk has been identified and the potential negative impact of our honeypots was limited using safeguards (rate limiter and intrusion prevention system). Also, each honeypot was deployed for a bounded period of time in our experiments (3 hours) in order to restrain its use by an attacker. This process was submitted to and validated by our institutional ethics review board.

VIII. Conclusion

HiFiPot is a MIPS-specific embedded devices emulation framework adapted for cyber deception. Our work was guided by the dual objectives of emulating many different devices while remaining hidden from an attacker. We defined an iterative procedure to correct encountered problems with minimal modifications during emulation. This procedure triggers several purpose-specific *Enhancers*, as for example for enabling Internet connectivity, allowing the use of NVRAM, etc.

Our evaluation has shown that HiFiPot has a higher capability to emulate different devices in comparison to FirmAE [10]. Our solution emulated successively more than 500 embedded devices in honeypot-mode and the results show that HiFiPot can capture human-based attacks and recent malware binaries. However, a large-scale deployment is planned and would rely on multiple servers with non institutional-related IP addresses to avoid disclosing ourselves.

[6]Available at https://honeyscore.shodan.io/, last accessed the 10/19/2022

REFERENCES

[1] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium*, 2017.

[2] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX annual technical conference, FREENIX Track*, 2005.

[3] Daming Chen, Manuel Egele, Maverick Woo, and David Brumley. Towards automated dynamic analysis for linux-based embedded firmware. 01 2016.

[4] Javier Franco, Ahmet Aris, Berk Canberk, and A. Selcuk Uluagac. A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. *CoRR*, abs/2108.02287, 2021.

[5] Alireza Ghasempour. Internet of things in smart grid: Architecture, applications, services, key technologies, and challenges. *Inventions*, 4(1), 2019.

[6] Juan David Guarnizo, Amit Tambe, Suman Sankar Bhunia, Martin Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. Siphon: Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, CPSS '17, page 57–68, New York, NY, USA, 2017. Association for Computing Machinery.

[7] Muhammad A. Hakim, Hidayet Aksu, A. Selcuk Uluagac, and Kemal Akkaya. U-pot: A honeypot framework for upnp-based iot devices. In *IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, 2018.

[8] Junaid Haseeb, Masood Mansoori, and Ian Welch. A measurement study of iot-based attacks using iot kill chain. In *IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020.

[9] Pierre-Marie Junges, Jérôme François, and Olivier Festor. Software-based Analysis of the Security By Design in Embedded Devices. In *IFIP/IEEE International Symposium on Integrated Network Management*, Bordeaux, France, 2021.

[10] Mingeun Kim, Dongkwan Kim, Eunsoo Kim, Suryeon Kim, Yeongjin Jang, and Yongdae Kim. Firmae: Towards large-scale emulation of iot firmware for dynamic analysis. In *Annual Computer Security Applications Conference*, ACSAC '20, page 733–745, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the iot: Mirai and other botnets. 50:80–84, 01 2017.

[12] Efrén López-Morales, Carlos Rubio-Medrano, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang, Tiffany Bao, and Gail-Joon Ahn. Honeyplc: A next-generation honeypot for industrial control systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20. ACM, 2020.

[13] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. Iotcandyjar : Towards an intelligent-interaction honeypot for iot devices. 2017.

[14] Vincent Nicomette, Mohamed Kaâniche, Eric Alata, and Matthieu Herrb. Set-up and deployment of a high-interaction honeypot: experiment and lessons learned. *Journal in Computer Virology*, 7(2):143–157, May 2011.

[15] Michel Oosterhof. Cowrie, a medium interaction ssh ad telnet honeypot. Available at https://www.cowrie.org/.

[16] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: Analysing the rise of iot compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., August 2015. USENIX Association.

[17] Niels Provos. A virtual honeypot framework. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 1–1, Berkeley, CA, USA, 2004. USENIX Association.

[18] Lukas Rist. Glastopf, a python web application honeypot. Available at https://github.com/mushorg/glastopf.

[19] Amit Tambe, Yan Lin Aung, Ragav Sridharan, Martín Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. *Detection of Threats to IoT Devices Using Scalable VPN-Forwarded Honeypots*, page 85–96. Association for Computing Machinery, New York, NY, USA, 2019.

[20] Upi Tamminen. Kippo, a medium interaction ssh honeypot. Available at https://www.honeynet.org/projects/old/kippo/.

[21] Kevin Timm. Honeyweb, a python http server honeypot. Available at http://www.citi.umich.edu/u/provos/honeyd/contrib/ktimm/.

[22] Emmanouil Vasilomanolakis, Shankar Karuppayah, Mathias Fischer, Max Mühlhäuser, Mihai Plasoianu, Lars Pandikow, and Wulf Pfeiffer. This network is infected: Hostage - a low-interaction honeypot for mobile devices. In *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, SPSM '13. ACM, 2013.

[23] Alexander Vetterl and Richard Clayton. Honware: A virtual honeypot framework for capturing cpe and iot zero days. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–13, 2019.

[24] Gérard Wagener, Radu State, Alexandre Dulaunoy, and Thomas Engel. Self adaptive high interaction honeypots driven by game theory. In Rachid Guerraoui and Franck Petit, editors, *Stabilization, Safety, and Security of Distributed Systems*, Berlin, Heidelberg, 2009. Springer.

[25] Peter Weidenbach and Johannes vom Dorp. Home router security report 2020. Technical report, Fraunhofer Institute for Communication, Information Processing and Ergonomics, 2020.

[26] Moeka Yamamoto, Shohei Kakei, and Shoichi Saito. Firmpot: A framework for intelligent-interaction honeypots using firmware of iot devices. In *2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 405–411, 2021.

[27] Armin Ziaie Tabari and Xinming Ou. A multi-phased multi-faceted iot honeypot ecosystem. In *SIGSAC Conference on Computer and Communications Security*, CCS '20. ACM, 2020.

[28] Haris Šemić and Sasa Mrdovic. Iot honeypot: A multi-component solution for handling manual and mirai-based attacks. In *2017 25th Telecommunication Forum (TELFOR)*, pages 1–4, 2017.