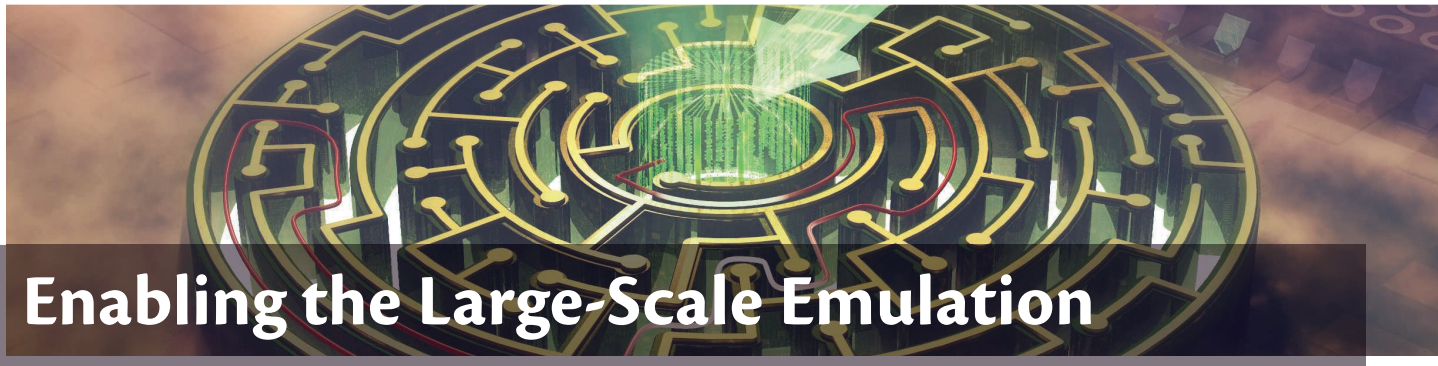


Enabling the Large-Scale Emulation of Internet of Things Firmware With Heuristic Workarounds

Dongkwan Kim and Eunsoo Kim | Korea Advanced Institute of Science and Technology
Mingeun Kim | Affiliated Institute of the Electronics and Telecommunications Research Institute
Yeongjin Jang | Oregon State University
Yongdae Kim | Korea Advanced Institute of Science and Technology

To evaluate Internet of Things device security, researchers have attempted to emulate and dynamically analyze firmware. However, this approach cannot deal with complex hardware/environmental diversities. We show that heuristic workarounds can enable firmware emulation and facilitate the discovery of vulnerabilities.

Billions of Internet of Things (IoT) devices are among us, from smart speakers to Internet-connected power outlets and light bulbs. Because they are always connected to the Internet, it is critical to discover any security vulnerabilities they might have. Although IoT products are simple and small, recent distributed denial-of-service (DDoS) attacks, which originate from a massive number of the devices, have demonstrated that malicious actions are critical threats. The attacks can generate traffic volume of more than 1 Tb/s, which can shut down important Internet services, such as DynDNS (in 2016)[1] and GitHub (in 2018).[2]

Scaling up vulnerability analysis is the key to neutralize security threats in devices and convoluted IoT ecosystems that consist of numerous manufacturers, products, and applications, among others. Applying recently advanced dynamic security analysis, such as fuzzing and automated pentesting, to IoT firmware on the elastic cloud could facilitate the approach. To do so, particularly for running IoT firmware on the cloud, we require an emulation framework. To this end, research projects have approached this problem as a hardware emulation challenge; that is, mimicking hardware and peripheral devices of the IoT ecosystem to make the replicated environment as precise as the real one. Based on this approach, Firmadyne,[3] the state-of-the-art firmware emulation framework, was designed for large-scale analysis of Linux-based IoT devices. Specifically, it leverages a customized Linux kernel and libraries to emulate hardware peripheral devices, such as a flash memory referred to as *nonvolatile random-access memory* (*NVRAM*).

Nonetheless, the hardware emulation approach is not a silver bullet in practice because building an

Digital Object Identifier 10.1109/MSEC.2021.3076226
Date of current version: 14 May 2021

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

emulation environment for supporting numerous IoT devices is challenging by itself. Each IoT device is accompanied by a specific set of peripheral hardware from a plethora of manufacturers. Further, IoT firmware images often rely on several configuration vectors that match specific hardware. Under such circumstances, for successful emulation, one should enumerate and virtualize specific hardware peripherals and runtime environments to mimic device behavior correctly. However, doing so is extremely challenging, owing to the complex diversity in IoT hardware and the associated implementation practices. Thus, Firmadyne can emulate only 183 of 1,124 (16.3%) firmware images in wireless routers and Internet Protocol (IP) cameras, which we collected from the top eight vendors.

Systematizing Heuristic Workarounds Toward Realizing Large-Scale Emulation

The low success rate of the hardware emulation approach implies that building an environment as precise as the real one may not be the only way to run firmware for dynamic security analysis. To counter the hardware complexity of the IoT ecosystem, we propose that well-systematized heuristic workarounds could be an alternative approach to achieve a better firmware emulation success rate in practice. By analyzing the numerous Firmadyne emulation failure cases, we observed that simple changes in device/software configurations could enable firmware continue to run by preempting failures that could arise by adopting the hardware emulation approach. By systematizing such heuristic workarounds and applying them as plug-ins, the system that we developed, FirmAE,[4] improved the number of firmware emulation success cases from 183 (16.3%) to 892 (79.4%). With the increased success rate, the system also discovered 306 (≈23 times, compared to Firmadyne) more vulnerabilities by applying dynamic security analysis techniques, such as fuzzing and one-day exploit testing.

Enabling Large-Scale Emulation

This article 1) suggests the possibility of systematizing well-developed heuristics as an approach to enable large-scale firmware emulation in practice and 2) summarizes how we discover and systematize such heuristic workarounds.

Background: IoT Firmware Analysis

An IoT device is often built for a specific purpose; examples include wireless routers, IP cameras, smart speakers, and so on. Thus, these devices are composed of specialized hardware peripherals and software to meet their intended purposes. Such hardware is controlled by firmware, which consists of a custom set of bootloaders, operating system kernels, and filesystems, that consists of tools and programs to accomplish required jobs. IoT devices often communicate over the Internet to the cloud and to mobile phones to provide user control, typically via a web interface. In summary, IoT devices are specialized embedded computer systems, and, thus, dynamic analysis methodologies for them (as illustrated in Figure 1) are slightly different from those for general desktop/server programs.

Acquiring Firmware Images

Firmware images can be obtained directly from a device; however, such an approach requires a particular interface that is available only to manufacturers to prevent unintentional firmware access. Instead, firmware can be updated via manufacturers' websites, and there are also several third-party servers that archive released firmware images. The images can also be automatically gathered via scraping tools, such as Scrapy.[5] Firmware images are usually packed and need to be unpacked for analysis. An image typically includes a bootloader, kernel, and filesystem that contains the applications and tools that are necessary for a device. To unpack firmware images, one can utilize tools such as Binwalk.[6] From a given image, this tool scans predefined signatures for diverse types of

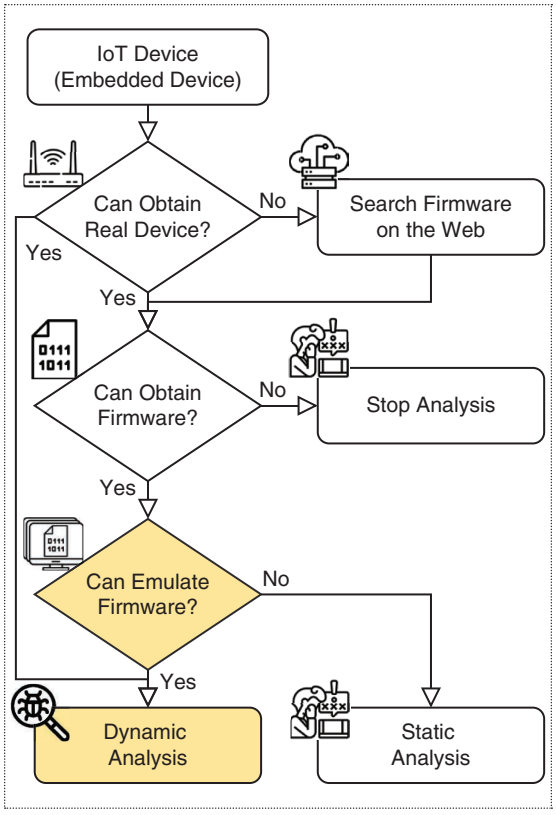


Figure 1. The typical IoT device analysis procedure. The colored logic indicates our focus.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

files. If it finds a match, it extracts the file from the image and continues the scan.

Emulating Device Firmware

After obtaining the files from a firmware image, one can apply static or dynamic analysis. Owing to the absence of runtime information, static analysis often produces numerous false positives. In contrast, dynamic analysis directly runs target programs, thereby leading to fewer false positives. To apply dynamic security analysis, one needs to either 1) have a real device to run and control the firmware or 2) construct an emulated environment for the device runtime. The latter approach does not require real devices, and thus it enables large-scale dynamic analysis, preferably on elastic cloud services. The system that conducts emulation is denoted as the *host system*, and the emulated target system is referred to as the *guest system*.

Figure 2 details a typical firmware emulation procedure for dynamic analysis. After unpacking a firmware image, the emulation framework attempts to boot, i.e., run a bootloader and kernel, from the extracted filesystem. Next, the guest system boots, runs initialization steps, and configures the network functionality. Finally, the guest system runs applications, such as web servers and common gateway interface programs, that interact with libraries and device drivers in the emulated system.

Dynamically Analyzing Emulated Firmware

After successfully running all necessary programs in firmware, dynamic analysis can be applied, particularly for discovering potential vulnerabilities of a target IoT device. Popular methods for such dynamic security analysis are 1) applying advanced fuzzing techniques, such as American fuzzy lop,[7] and 2) applying manual/automated pentesting based on known vulnerabilities by using tools such as Metasploit[8] and RouterSploit.[9]

Challenges to Firmware Emulation

Despite several benefits, such as the capability for dynamic analysis and exploiting the scalability of the elastic cloud for testing, IoT firmware emulation is extremely challenging. The difficulty mainly originates from the inconsistencies between the real and emulated environments. Resolving such inconsistencies is not a trivial task, owing to the convoluted IoT ecosystem. The complexity comes from having wide diversity in device and hardware manufacturing and from having no standardized software development practices. For instance, each IoT device has different hardware peripherals, such as cameras, flash memories, and sensors, and they work closely with the kernel and applications in the IoT firmware. Without dealing with requests to these peripherals during emulation, the kernel and applications in the emulated firmware may crash, and no further emulation or dynamic analysis can proceed.

Firmadyne (Automatic Emulation Framework)

Research projects approach the difficulties as a hardware emulation challenge, i.e., mimicking IoT ecosystem hardware and its peripheral devices so that the replicated environment becomes as precise as the real one. Among such frameworks, Firmadyne[3] is the current state of the art (succeeded by FirmAE[4]), designed for large-scale analysis of Linux-based IoT devices. It leverages a customized Linux kernel and libraries to emulate hardware and peripheral devices, such the NVRAM.

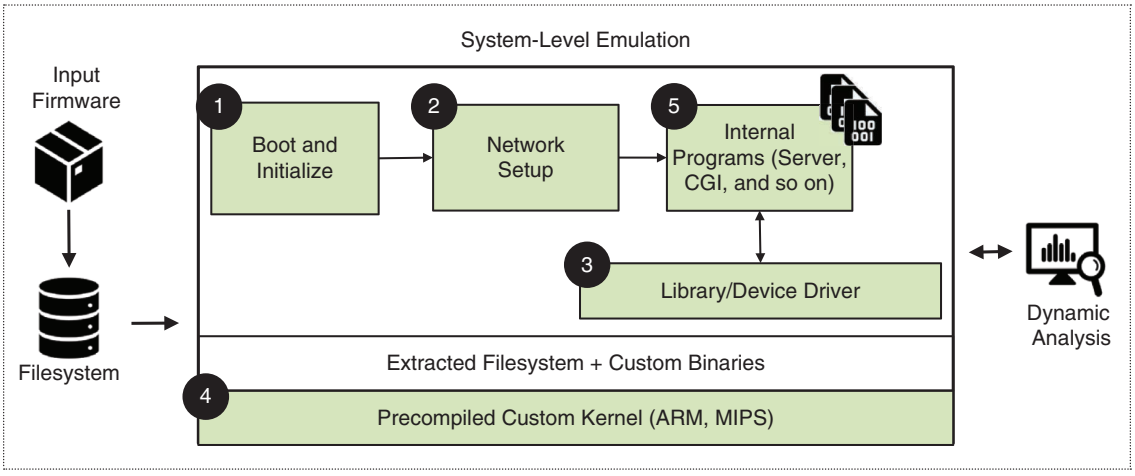


Figure 2. The typical firmware emulation procedure for dynamic analysis. The colored components indicate where emulation problems may occur. CGI: common gateway interface.

thorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

Limitations of Firmadyne

Although Firmadyne's precise hardware emulation appears promising, its success rate, in reality, is low. Among 1,124 IoT firmware images that we collected from the top eight vendors of wireless routers and IP cameras, Firmadyne can emulate only 330 (29.4%) for networking functionality. For running the applications (e.g., web servers) in the firmware, Firmadyne demonstrates a much worse success rate. More specifically, it can emulate web applications from only 183 (16.3%) firmware images. Such poor success can hinder the application of dynamic analysis to numerous IoT devices.

Toward Enabling Large-Scale Firmware Emulation for Security Testing

For dynamic software security testing, such as applying human pentesting and fuzzing, perfect hardware compatibility at the emulation layer is not required. Instead, the process requires only that the minimum requirement to properly run applications in firmware be met. Notably, we can achieve that without having perfect emulation of the hardware. For example, for a wireless router, we need to 1) boot the firmware operating system, 2) set up the network interfaces and required connections, and 3) run a web server to provide an administrative interface to the router. However, there could be several factors that might not be achievable by general emulators, such as QEMU. These factors include 1) the absence of correct values in the NVRAM (e.g., boot parameters) and missing devices (e.g., some hardware devices required for booting) to fulfill the boot condition, 2) a lack of network interfaces (e.g., different network interface controllers) and connections (missing the Internet or intranet connection) required to communicate with the applications in the emulated system, and 3) a lack of conditions to launch the web server with the correct configuration, such as the server IP address and port.

Unlike prior approaches that aim to emulate hardware behaviors so that firmware applications function correctly, we take a different perspective. That is, instead of resolving all hardware emulation dependencies, we aim to build an abstraction environment that satisfies the minimum requirements to execute booting, initialization steps, and applications to make a device available for dynamic security testing. Specifically, the following two properties illustrate our abstraction emulation goal for dynamic security testing on IoT devices:

1. *Network reachability*: The emulated network should be reachable from the host system.
2. *Service availability*: An emulated program should be available for dynamic analysis.

Achieving these goals may not address the fundamental emulation problem, i.e., exactly mimicking device behaviors. However, we believe that learning heuristics and systematizing such knowledge to create an abstraction environment is sufficient to set up the networking functionality and run applications in firmware, although the environment is not identical to the target device. The environment with various heuristics runs core features of target IoT devices, interacts with the applications in firmware, and facilitates dynamic security testing. We emphasize that the key idea is to deal with high-level properties to meet the requirements for firmware applications and not to accurately emulate the underlying hardware. Since such heuristics deal with high-level properties rather than hardware problems, they can be transferred across devices and may preempt failures, even with different root causes. In the following, we present a simple example of how heuristics can run applications in firmware by avoiding a hardware problem that is irrelevant to dynamic security testing.

Example: LED Failure in a Wireless Router

Consider the example of running the firmware of a wireless router. Wireless routers often have LEDs that indicate their runtime status, and the lights are not crucial for the dynamic security testing of functionality (i.e., finding remote code execution and cross-site scripting vulnerabilities). However, the device initialization process may crash if the process cannot set up the LEDs, i.e., if the emulated environment does not give nonerror responses to the application. Owing to this, the initialization process stops and does not make progress on configuring other critical parts, such as setting up the network and running web servers. Consequently, the emulation fails with the LED error, and thus dynamic security testing cannot be applied. As a counterargument to this example, a simple heuristic, which is independent of the stopped initialization step, can enable the system to set up the network. By doing so, the other nonerror-related parts of the firmware, such as the web server, can successfully run (if the network is correctly configured); then, one can apply dynamic security testing to the web interface.

Wireless Routers as an IoT Firmware Case Study

We run wireless router firmware for a large-scale evaluation of the IoT firmware emulation capability of our approach, making an abstraction emulation layer with systematized heuristics. This is because 1) most wireless router firmware requires networking and web server functionality, which is mandated by most IoT devices in general; 2) wireless routers were introduced

thorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

in the early 2000s and are highly diversified in their models and hardware/software configurations; and 3) their firmware is available for large-scale analysis (we collected 1,079 images). Furthermore, wireless routers are crucial to IoT and home security because they are the gateway to home networks and can manage other devices via internal networking. Therefore, we focus on emulating wireless routers for testing our hypothesis about systematized heuristics for emulation.

Systematizing Heuristics From Failures

As the first step to learn and systematize heuristics that avoid emulation failures, we investigate cases of Firmadyne[3] firmware emulation failure. We collected 1,079 wireless router images from the websites of the top eight vendors.[10] Among the images, we emulated 526 old versions by using Firmadyne; the tool could succeed with only 16.9%. For the results, we categorized the identified failure cases by the place where the emulation failed (see Figure 2). In the process of analyzing the failures, we were able to systematize several heuristics that can address and avoid similar shortcomings. Note that some of the heuristics have also been proposed in previous approaches.[3,11] We introduce examples of such heuristics, which are described in Table 1. For more detailed information, please refer to the technical version of this article.[4]

Heuristics for Handling Boot Failures

The booting procedure includes executing several programs that initialize the system environment, and it may fail if the emulated environment cannot meet the requirement for executing any (or even a part) of the programs that are required. This would result in an emulation getting stuck, such as in the case of kernel panic. Many boot failures were observed to encounter a kernel panic. By analyzing these cases, we identified two kinds of problems in boot emulation. On the one hand, the emulator kernel, which is different from the kernel that runs on the device, failed to find the correct initializing program, which is custom configured by the device manufacturer. Although program paths can differ depending on firmware images, the kernel image used in the emulator searches only the predefined paths, such as `/sbin/init` and `/etc/init`. As a result, programs on a different path, such as `/etc/preinit`, cannot be properly executed. On the other hand, the boot process fails by not having files and directories required by the `init` program. If the program accesses such paths that do not exist, it crashes and eventually halts the booting process.

To systematize heuristics, we addressed the first problem using the original kernel in the target device firmware. A firmware image typically consists of a kernel image and programs/data in a filesystem. Specifically, we searched the kernel image for the string literals accessed by the kernel with the target kernel configuration. These strings are predefined by the device manufacturer in the development stage and are naturally embedded in the kernel image. An example of these heuristics is that we searched the string "`init`" in the target kernel and obtained a string of `console=ttyS0,115200 root=31:08 rootfstype=squashfs init=/etc/preinit`, which seems to be a kernel boot argument. From this string, we could identify that the initializing program is located at `/etc/preinit` and boot the kernel appropriately.

Table 1. Examples of heuristics to address emulation problems for running web services in wireless routers.

Where	Emulation problem	Heuristics
❶ Boot	Improper booting sequence	Use the booting sequence of the original kernel
	Missing files and directories	Prepare files and directories before the emulation
❷ Network	No support for IP aliasing and VLAN	Fix routing rules and network interface settings
	No network interface	Forcibly set up default network interface
❸ Library	Unknown NVRAM values	Search key value pairs from the filesystem
	Invalid return of NULL values	Return a valid string pointer instead of NULL
❹ Kernel	Insufficient support of kernel modules	Emulate functions, such as ioctl, by using libraries
	Improper kernel version	Upgrade the kernel version to 4.1
❺ Programs	Unexecuted web servers	Forcibly execute the web server
	No support for extra commands	Add full-featured BusyBox

VLAN: virtual local area network.

thorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

Similarly, we applied a string literal search strategy to the firmware filesystem to address the second problem. As a preprocessing stage to boot a firmware image, we extracted strings that were highly likely to indicate program/directory path names, from the programs in the firmware filesystem. More specifically, we obtained several strings that started with general Unix system paths, such as `/var` and `/etc`. Then, we placed directories or files based on those path values. As a result, we could successfully boot many firmware images without errors.

Heuristics for Handling Network Failures

The next step is to develop heuristics for handling network failures. After completing the booting procedure, the network must be set up correctly so that the emulation host can communicate with the emulation guest—the running firmware. Any failure in the network setup will result in a failure to run dynamic analysis, even though internal programs, such as web services, are functioning correctly; this is because the host system cannot interact without networking functionality.

To this end, we observed several networking failures in emulating firmware images. First, existing emulation frameworks cannot handle important network operations widely used in wireless routers, such as IP aliasing and virtual local area networks (VLANs). IP aliasing enables assigning multiple IP addresses to a single network interface, and VLANs facilitate logically grouping the network; both features are widespread in modern wireless routers. To handle such functionalities, we developed a dedicated routine that automatically configures routing rules and interface settings for IP aliasing and VLANs, and we applied it to the emulation runtime.

Furthermore, we analyzed cases where the existing emulation frameworks failed to retrieve any information about network interfaces for some firmware images. We also deduced that these cases originated from a failure in boot procedure before attaining the network setup part; recall the example case of the LEDs. To resolve this issue, we developed a heuristic that compels the emulated system to set up a default network interface (e.g., eth0, a Realtek device) in a manner similar to the previous approach,[11] thereby avoiding such failure cases.

Heuristics for Handling NVRAM Failures

NVRAM is a type of flash memory, which works as a simple key/value storage. It stores various information for running target IoT devices, such as configurations of a device as well as peripheral equipment. The NVRAM itself is a popular peripheral device for IoT devices, including wireless routers, because it is equipped with various peripheral hardware. In essence, the information stored in the NVRAM is required to properly operate peripherals and a target device. Because the internal programs in firmware store/fetch configuration values to/from it, the NVRAM has to be emulated correctly. Internal programs in IoT firmware often interact with the NVRAM via libraries. To exploit this feature, other emulation frameworks built an additional library that mimics the interaction with the NVRAM to run the firmware without actually having the real NVRAM.

For example, in Firmadyne,[3] a custom library, `libnvram`, is implemented to emulate the NVRAM. It is loaded on top of other libraries to make internal programs utilize it instead of interacting with the real NVRAM. It initializes its contents with a hard-coded list of default files and their values; such files are common in many IoT devices for the factory reset functionality. For unknown keys, it naively returned the `NULL` value. However, such an approach to utilize a hard-coded list was insufficient to cover diverse devices, resulting in many failure cases.

To resolve such issues, we developed a heuristic strategy that automatically searches the firmware filesystem for the requested values. Specifically, we first ran the firmware in our emulation environment to record all fetched (required) keys during the first stage of emulation, which would likely fail. Then, we scanned the firmware filesystem to find the files that contained the recorded keys and then fetched the corresponding values from those files. By performing this interactively, we could fetch most of the required keys for the NVRAM of the target device.

The heuristic may fail if we cannot successfully find the matching values for the requested keys to the NVRAM. To handle such unknown key/value pairs, we extended the custom library to return a valid pointer, which indicates to an empty string instead of returning a `NULL` pointer as a value. This heuristic is based on the fact that many programs supply the returned value, which is most likely a configuration value as a string, into string-related functions, such as `strcpy()` and `strtok()`. Hence, a `NULL` value results in the program crashing immediately, while returning a zero-length valid string will pass such function executions successfully. As a result, this strategy significantly decreased NVRAM-related crashes and enabled programs to run appropriately, even without correct configuration values.

Heuristics for Handling Failures in the Kernel

In addition to the NVRAM, internal programs can cooperate with peripheral devices through kernel

thorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

modules, i.e., device drivers. In particular, if a kernel or module version does not match the real device, the programs cannot interact with peripheral equipment and may crash, resulting in the emulation failure. We analyzed such failures and found that Firmadyne uses kernel version `2.6.32`, which does not support recent features used in the target device and its firmware. In this case, upgrading the kernel version to a newer one, e.g., `v4.1.17`, resolved these issues for most cases and emulated more firmware images.

In addition to kernel version issues, firmware emulation may fail if kernel drivers cannot communicate with programs in firmware. This problem is similar to that of the NVRAM; in other words, the device driver should return corresponding values to specific requests. In the case of Firmadyne, the issue was dealt in a manner similar to the NVRAM. That is, a mimicking kernel module was implemented; it emulated the `ioctl` interaction between kernel modules in firmware and peripheral devices. However, such an approach generates many emulation failures; the values, which are passed to and returned from the call, significantly vary depending on the firmware as well as the device architecture. To resolve this, we developed a heuristic strategy that adds functions returning predefined values, regardless of the `ioctl` interface/parameter variant. This heuristic let the programs in firmware continue the execution without having system call errors.

Heuristics for Handling Application Failures

Apart from the booting, networking, NVRAM, and kernel, application execution can be disturbed during the emulation. Running applications is the most critical step in dynamic security testing because applications in firmware contain a device's core logic, which is the actual target of security testing. In emulating applications in firmware, specifically for web interfaces, we discovered several other problems that obstruct execution. First, the web server application failed to run even with a successful network setup in some images. We expect that the network device is set up after running the web server during emulation; hence, the web server failed to find and bind to the network device. In such a case, forcing the web server to run after finishing the entire initialization step could address the problem.

Additionally, missing files required for the emulation environment in the firmware filesystem can lead to emulation failure. The emulated environment may not contain these files because many IoT device developers remove programs that the target device would never use so that they can reduce the storage size. However, in the emulation environment and in applying systematized heuristics, we might need several configuration tools, such as `ifconfig`, `ip`, and so on. Missing such tools can result in emulation failures, and the applied heuristics can be ineffective. To resolve this, we added the latest version of `BusyBox`, a Swiss army knife in the Unix box, to supply required command line tools to the emulation environment.

FirmAE: Systematizing Heuristics Learned

We systematize heuristics learned from the emulation failure case analysis with FirmAE, which is an automated emulation framework for large-scale dynamic analysis of IoT firmware. Figure 3 presents a component-wise

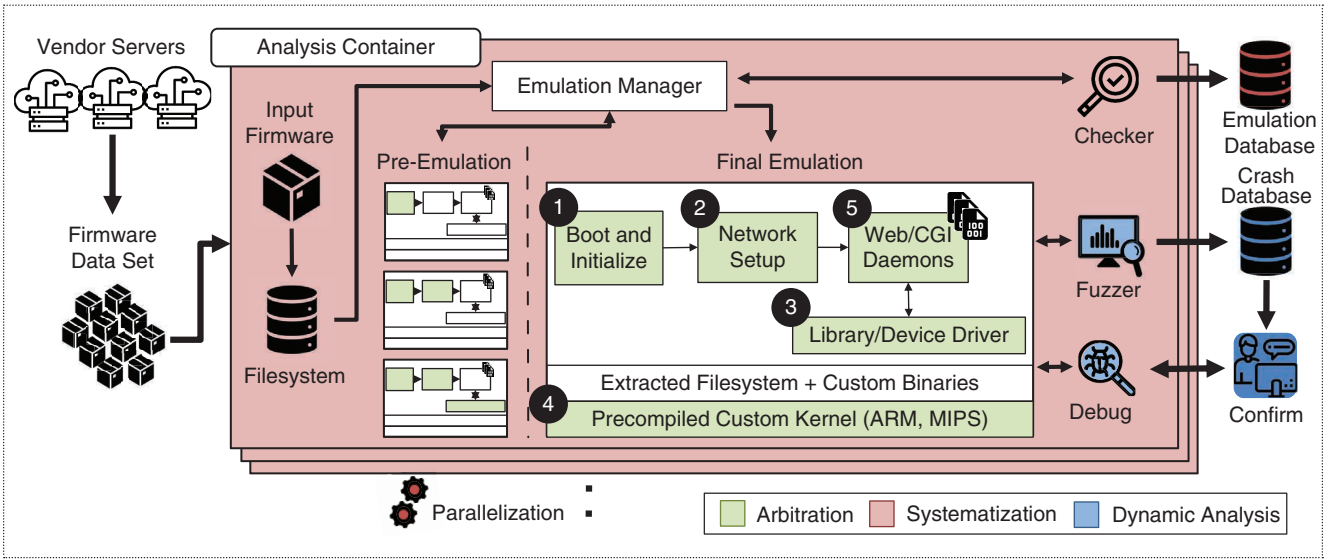


Figure 3. An overview of the FirmAE architecture with dynamic analysis.

thorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

overview of FirmAE. The key difference of FirmAE is that in addition to the techniques developed by Firmadyne, FirmAE applies various systematized heuristics to increase the emulation success rate. From steps 1–5, FirmAE applies corresponding heuristics to preempt any detected emulation failures. In the following, we evaluate FirmAE for its emulation and dynamic security testing ability.

Effectiveness in IoT Firmware Emulation

To evaluate emulation effectiveness, we ran 1,079 wireless router firmware images from the top eight vendors (based on device popularity) on FirmAE. With these images, we tested two hypotheses:

- *H1*: How successfully do well-systematized heuristics emulate firmware images when compared to the existing framework?
- *H2*: Are the heuristics learned from old firmware images transferable to newer firmware versions?

We split the entire firmware set in two: `AnalysisSet` and `LatestSet`. We used `AnalysisSet`, having 526 old images, to observe failures and develop heuristics. Then, we evaluated those heuristics on the other data set, consisting of 553 images, including only the latest ones. This setup will test the hypothesis that the heuristics learned from old images (`AnalysisSet`) are transferrable to newer ones (`LatestSet`). Additionally, we ran IP camera firmware images to test the third hypothesis:

- *H3*: Are the heuristics transferrable to IoT devices other than wireless routers?

In particular, we collected 45 of the latest images of IP cameras (`CamSet`). The data sets have no intersection; i.e., they do not share any images. We compared the number of successful emulations using FirmAE and Firmadyne for each data set, and the final results are in Figure 4. From `AnalysisSet`, FirmAE could successfully emulate 483 images, while Firmadyne could emulate 89 (supporting H1). For testing the transferability of heuristics to newer versions, from `LatestSet`, FirmAE could successfully emulate 382 images, while Firmadyne could emulate 92 (supporting H1 and H2). For testing the transferability of heuristics to a different class of IoT devices, from `CamSet`, FirmAE could successfully emulate 27 IP camera images, while Firmadyne emulated two (supporting H3). The results not only support our hypotheses but demonstrate that the emulation success rate (79.36%) significantly increased from that of Firmadyne (16.28%), supporting H1.

Dynamic Analysis Capabilities

FirmAE relies on an imperfect emulation of firmware devices based on heuristics from empirical observation. Therefore, its capability of applying dynamic security analysis would be questionable, i.e., whether FirmAE can be used for discovering security vulnerabilities or not. To demonstrate that our heuristics-based emulation approach is indeed effective, we tested the following two hypotheses:

- *H4*: Can known vulnerability (i.e., one-day) exploits work against firmware running on FirmAE?
- *H5*: Can dynamic analysis on FirmAE discover new vulnerabilities (i.e., zero-days)?

To test H4, we launched one-day exploits from RouterSploit[9] to firmware images on FirmAE and Firmadyne, respectively. Table 2 includes the results. While only 14 exploits worked against Firmadyne,

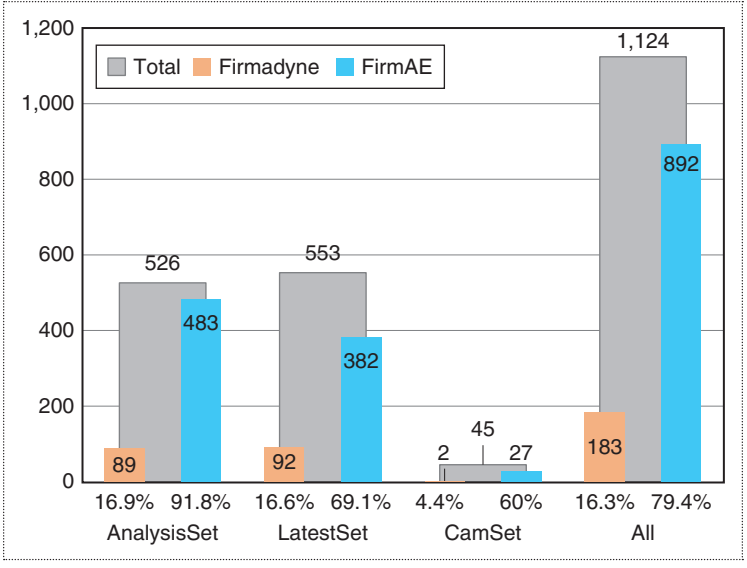


Figure 4. The number of emulated firmware images obtained by running FirmAE and Firmadyne on each data set.

Table 2. The number of one-days discovered on the outdated firmware images (`AnalysisSet`).

Vulnerability	Firmadyne	FirmAE
Information leak	0	17
Command injection	10	152
Password disclosure	4	146
Authentication bypass	0	5
Total	14	320

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

320 one-day exploits worked against FirmAE (supporting H4). The implication of supporting H4 is that if one-day exploits work against the firmware image running on FirmAE, then dynamic analysis applied to FirmAE can discover such vulnerabilities from the emulation; hence, the dynamic analysis applied to FirmAE can discover significantly more vulnerabilities than Firmadyne.

To test H5, we ran images from `LatestSet` and `CamSet`, consisting of only the latest firmware images, to check if FirmAE can help discover new vulnerabilities via dynamic security testing. In this regard, we define vulnerabilities as 1) those that are known but unpatched on the latest device versions and different models (one-days) and 2) new (zero-days). Table 3 lists the unique number of newly identified vulnerabilities, affected devices, and vendors. First, we launched one-day exploits from RouterSploit[9] via the same approach described previously and discovered 11 one-day vulnerabilities affecting 72 unique devices (supporting H4). To discover zero-day vulnerabilities, we implemented a simple fuzzer that injected input to the web interface. By using the fuzzer, FirmAE successfully identified 12 new zero-day vulnerabilities affecting 23 unique devices (supporting H5). For more details, please refer to the technical version of this article.[4] In summary, the dynamic analysis results demonstrate that the heuristic-based emulation approach of FirmAE is effective for vulnerability analysis.

Responsible Disclosure Zero-Day Vulnerabilities

We reported all discovered zero-day vulnerabilities to the corresponding vendors, and these were acknowledged by December 2019.

Table 3. The number of new vulnerabilities discovered on the latest firmware images (`LatestSet` and `CamSet`).

Type	Vulnerability	Number of vulnerabilities	Number of devices	Number of vendors
One-day	Information leak	2	32	2
	Command injection	5	28	2
	Backdoor	2	3	1
	Path traversal	2	9	2
Zero-day	Command injection	7	16	2
	Buffer overflow	5	7	4
Total		23	95	6

Generality of the Proposed Heuristics

Although our heuristics significantly increased the emulation success rate and facilitated the discovery of vulnerabilities, they might not be applicable to new types of devices and configurations because of the hardware/environmental diversities in the IoT ecosystem. Since we designed our heuristics by empirically analyzing the failure cases of our firmware data set, new types of devices may require new heuristics to deal with their failure cases. However, as shown with H2 and H3, the developed heuristics can be transferred to newer device versions and similar device families. In this regard, we believe that further empirical investigation to develop additional heuristics is indispensable to handle the convoluted nature of the IoT ecosystem.

Toward Large-Scale Firmware Emulation

Recent advances in dynamic security testing, such as automated pentesting, fuzzing, symbolic execution, and their combination, can automatically discover security vulnerabilities in a scalable manner. Applying such dynamic analysis to the IoT ecosystem may improve security, especially by harnessing testing scalability to deal with numerous devices. However, the emulation of device firmware is challenging owing to the convoluted nature of IoT device hardware/software implementation practices.

Several approaches,[12–14] in addition to Firmadyne,[3] have attempted to precisely emulate hardware devices by modeling memory-mapped input/output operations in peripheral communication[12,13] and by building an abstract layer to deal with hardware emulations.[14] These methods are essential in the long run to achieve better accuracy in testing; however, such frameworks still suffer from limitations related to covering firmware across highly diversified IoT devices.

We believe that accumulating heuristic knowledge for workaround firmware emulation failures is the last-mile effort to overcome such limitations. Systematizing the heuristics learned from failure cases can enable large-scale firmware emulation, as such heuristic knowledge is transferrable to newer device versions and similar product families. We recommend future studies to conduct more empirical investigations and systematize and share the obtained knowledge for scalable security analysis of IoT ecosystems. ■

Acknowledgments

We thank the anonymous reviewers for their thoughtful comments. This work was supported by an Institute of Information and Communications Technology Planning and Evaluation grant (2018-0-00831), which was funded by the government of Korea.

thorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.

References

1. L. H. Newman, "What we know about Friday's massive east coast internet outage," Wired, Oct. 2016. https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/ (accessed Feb. 28, 2021)
2. L. H. Newman, "GitHub survived the biggest DDoS attack ever recorded," Wired, Jan. 3, 2018. https://www.wired.com/story/github-ddos-memcached/ (accessed Feb. 28, 2021)
3. D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards automated dynamic analysis for linux-based embedded firmware," in *Proc. Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, Feb. 2016.
4. M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "FirmAE: Towards large-scale emulation of IoT firmware for dynamic analysis," in *Proc. Comput. Security Appl. Conf. (ACSAC)*, Dec. 2020, pp. 733–745.
5. R. Mitchell, *Web Scraping with Python: Collecting More Data from the Modern Web*. O'Reilly Media Inc., 2018.
6. C. Heffner, "Firmware analysis tool," GitHub, San Francisco, 2010. https://github.com/ReFirmLabs/binwalk
7. M. Zalewski, "American fuzzy lop (AFL)," lcamduf.coredump.cx, 2017. http://lcamtuf.coredump.cx/afl (accessed Feb. 28, 2021).
8. Rapid7, "Metasploit," Rapid7, 2009. https://www.metasploit.com
9. Threat9. "RouterSploit," GitHub, San Francisco, 2016. https://github.com/threat9/routersploit (accessed Feb. 28, 2021).
10. M. Wilson, "Global premium wireless routers market 2019 by manufacturers, regions, type and application, forecast to 2024," Analytical Research Cognizance, 2019.
11. A. Vetterl and R. Clayton, "Honware: A virtual honeypot framework for capturing CPE and IoT zero days," in *Proc. APWG Symp. Electronic Crime Research (eCrime)*, 2019, pp. 1–13.
12. E. Gustafson et al., "Toward the analysis of embedded firmware through automated re-hosting," in *Proc. 22th Int. Symp. Res Attacks, Intrusions and Defenses (RAID)*, Beijing, Sept. 2019, pp. 135–150.
13. B. Feng, A. Mera, and L. Lu, "P2IM: Scalable and Hardware-independent Firmware Testing via Automatic Peripheral Interface Modeling," in *Proc. 29th USENIX Security Symp. (Security)*, Boston, Aug. 2020, pp. 1237–1254.
14. A. A. Clements et al., "HALucinator: Firmware re-hosting through abstraction layer emulation," in *Proc. 29th USENIX Security Symp. (Security)*, Boston, Aug. 2020, pp. 1201–1218.
15. *Proc. 29th USENIX Security Symp. (Security)*, Boston, Aug. 2020.

Dongkwan Kim is a Ph.D. candidate in the School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, 34141, South Korea. His research interests include software, embedded/cyber-physical systems, cellular networks, and machine learning. Kim received an M.S. from the School of Electrical Engineering at Korea Advanced Institute of Science and Technology. Contact him at dkay@kaist.ac.kr.

Eunsoo Kim is a Ph.D. candidate in the Graduate School of Information Security, Korea Advanced Institute of Science and Technology, Daejeon, 34141, South Korea. His research interests include finding vulnerabilities in various software and embedded systems. Kim received an M.S. from the Graduate School of Information Security, Korea Advanced Institute of Science and Technology. Contact him at hahah@kaist.ac.kr.

Mingeun Kim is a researcher at the Affiliated Institute of the Electronics and Telecommunications Research Institute, Daejeon, 34129, South Korea. His research interests include firmware emulation and analysis of Internet of Things devices. Kim received an M.S. from the Graduate School of Information Security, Korea Advanced Institute of Science and Technology. Contact him at rla5072@nsr.re.kr.

Yeongjin Jang is an assistant professor of computer science in the College of Engineering, Oregon State University, Corvallis, Oregon, 97331, USA. His research interests include computer systems security, particularly, identifying and analyzing emerging attacks for building secure systems. Jang received a Ph.D. from the School of Computer Science at the Georgia Institute of Technology. Contact him at yeongjin.jang@oregonstate.edu.

Yongdae Kim is a professor in the School of Electrical Engineering and an affiliate professor of GSIS, Korea Advanced Institute of Science and Technology, Daejeon, 34141, South Korea. His research interests include security issues for various systems, such as cyberphysical systems, cellular networks, peer-to-peer systems, and embedded systems. He received a National Science Foundation CAREER Award and a McKnight Land-Grant Professorship Award from the University of Minnesota. He is a former Network and Distributed System Security Symposium Steering Committee member and associate editor of *ACM Transactions on Privacy and Security*. Kim received a Ph.D. from the Computer Science Department at the University of Southern California. Contact him at yongdaek@kaist.ac.kr.

thorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on July 03,2026 at 08:16:28 UTC from IEEE Xplore. Restrictions apply.