HADES-IoT: A Practical Host-Based Anomaly Detection System for IoT Devices

Dominik Breitenbacher
SUTD, Singapore
dbreitenbacher@gmail.com

Ivan Homoliak
SUTD, Singapore
ivan_homoliak@sutd.edu.sg

Yan Lin Aung
SUTD, Singapore
linaung_yan@sutd.edu.sg

Nils Ole Tippenhauer
CISPA, Germany
tippenhauer@cispa.saarland

Yuval Elovici
SUTD, Singapore
yuval_elovici@sutd.edu.sg

ABSTRACT

Internet of Things (IoT) devices have become ubiquitous and spread across many application domains including the industry, transportation, healthcare, and households. However, the proliferation of the IoT devices has raised the concerns about their security – many manufacturers focus only on the core functionality of their products due to short time to market and low cost pressures, while neglecting security aspects. Consequently, vulnerabilities are left untreated, allowing attackers to exploit IoT devices for various purposes, such as compromising privacy, recruiting devices into a botnet, or misusing devices to perform cryptocurrency mining. In this paper, we present a practical **H**ost-based **A**nomaly **DE**tection **S**ystem for **IoT** (HADES-IoT) as a novel last line of defense. HADES-IoT has proactive detection capabilities, provides tamper-proof resistance, and can be deployed on a wide range of Linux-based IoT devices. The main advantage of HADES-IoT is its low performance overhead, which makes it suitable for the IoT domain, where state-of-the-art approaches cannot be applied due to their high-performance demands. We deployed HADES-IoT on seven IoT devices and demonstrated 100% effectiveness in the detection of current IoT malware such as VPNFilter and IoTReaper; while on average, requiring only 5.5% of available memory and causing only a low CPU load.

CCS CONCEPTS

• **Security and Privacy** → **Malware and Its Mitigation**; **Intrusion Detection Systems**.

KEYWORDS

Intrusion Detection; System Call Interception; IoT; LKM

ACM Reference Format:
Dominik Breitenbacher, Ivan Homoliak, Yan Lin Aung, Nils Ole Tippenhauer, and Yuval Elovici. 2019. HADES-IoT: A Practical Host-Based Anomaly Detection System for IoT Devices. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3321705.3329847

AsiaCCS '19, July 9–12, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6752-3/19/07…$15.00
https://doi.org/10.1145/3321705.3329847

1 INTRODUCTION

In recent years, the number of IoT devices connected to the Internet reached seven billion [11] and is expected to grow. Gartner estimates that more than 20 billion IoT devices will be connected to the Internet by 2020 [12]. Nevertheless, the advent of the IoT has brought challenges in many areas, including data storage, maintenance, and particularly, privacy and security [14], [10], [13]. Many IoT devices are designed with a specific purpose in mind, so their software and hardware is solely chosen to satisfy the requirements on core functionalities, e.g., using just as fast processor as needed to meet certain real-time constraints. When it comes to security, the preference is put on short time to market and budget constraints at the expense of more expensive and possibly more comprehensive security solutions. Therefore, embedded technology results in a trade-off between cost and security [14]; however, real-time requirements, computing capabilities, and energy consumption are also part of this trade-off.

For these reasons, IoT devices are often released with serious vulnerabilities, and this issue is further exacerbated by the fact that IoT devices are, in many cases, exposed on the Internet, and thus easily reachable by attackers. Most IoT devices are exploited due to operations security (OPSEC) issues, such as the use of weak or default passwords [3], [17]; however, they are also exploited due to buffer overflow, command injection, etc., [18]. These practical examples raise the question of *how to achieve greater security in IoT devices while minimizing the requirements on cost and the utilization of computational resources?* One way to raise the bar against attackers is to improve preventive OPSEC countermeasures, such as employing strong passwords and conservative access control. However, in the case of more sophisticated attacks, IoT devices must be protected by other dedicated means, such as host-based intrusion detection systems or IoT-specific antivirus systems. Nevertheless, there might not be enough economic incentives for companies developing low-cost competitive IoT devices to invest in security countermeasures that are also computationally demanding.

Given the abovementioned constraints, we make the following fundamental observation: in contrast to general computing devices such as laptops or mobile devices, IoT devices have, by design, a well-defined and stable functionality. However, when an IoT device is compromised by malware, its behavior changes significantly. Therefore, intrusion detection and anomaly detection systems are promising options for securing IoT devices. Although, these systems

might be deployed outside of the device and perform inspection of network traffic [8], they might be evaded by various payload-based [6, 19] and non-payload-based obfuscations [4, 9]. Hence, we argue that behavioral changes are best observed from "within" a device, for instance, by monitoring which processes are running and what actions they perform. For this reason, we consider host-based behavior analysis as an effective last line of defense.

Proposed Approach. In this paper, we propose a lightweight Host-based Anomaly DEtection System for IoT devices (HADES-IoT) that monitors process spawning and stops any unauthorized program before its execution, thus providing proactive detection and prevention. To achieve real-time detection, we developed HADES-IoT as a loadable kernel module of Linux OS. Such a design decision allows us to make HADES-IoT tamper-proof against an attacker (with superuser privileges) trying to disable it. Since IoT devices are strictly resource constrained, we created HADES-IoT in a lightweight fashion, ensuring that primary functionality provided by the device is not affected. HADES-IoT supports various types of IoT devices, as most of them are based on the Linux OS [21].

Contributions: In summary, our contributions are as follows:

- We present a novel host-based anomaly detection and prevention approach that is based on whitelisting legitimate processes on an IoT device.
- We develop a proof-of-concept of our approach and evaluate its effectiveness on several IoT devices.
- We show that HADES-IoT is generic and can be easily adapted to any Linux kernel version.
- We demonstrate that HADES-IoT is tamper-proof and thus provides resilience against attackers focusing on disabling its protection mechanisms.

2 PROBLEM STATEMENT

The main objective of this work is to propose a security solution that protects the bulk of the existing IoT devices against remote exploitation of any vulnerabilities (including zero-day ones).

2.1 Assumptions

In this work we aim at Linux-based IoT devices. According to [21], the market share of Linux-based IoT devices is over 80%, and hence we cover the vast majority of existing IoT devices. Furthermore, we assume that (1) all of the executables installed on an IoT device are benign, and (2) an attacker does not tamper with the device either before or during the bootstrapping of our proposed approach. However, we assume that the default executables of an IoT device may contain a vulnerability enabling execution of arbitrary binaries – either binaries that already exist on a device or binaries delivered by an attacker. Finally, to the best of our knowledge, it is not common practice for manufacturers to enable security features, such as *SELinux*, *Auditd*, or *Access Control Lists* on their IoT devices mainly due to performance reasons. Therefore, we assume that our approach is the only security solution deployed on an IoT device.

2.2 Attacker Model

We assume that an IoT device is protected by our approach and is connected to the Internet. Therefore, an attacker is able to find it using a custom scanner or publicly available services such as *Shodan*. The attacker can do reconnaissance of an IoT device to reveal open ports and running network services. To make our attacker model challenging, we consider that an attacker is capable of exploiting any vulnerability (i.e., potentially zero-day) on an IoT device through one of the network services running on the device. We further consider that once an attacker "is inside" an IoT device, he is granted superuser privileges, since it is common for many IoT devices to have just a superuser account.

Device	Type	Kernel Version	CPU		Memory	
			Arch.	Speed	Total	Avail.
Netgear WNR2000v3	Router	2.6.15	MIPS	265.2	32	16.06
ASUS RT-N16	Router	2.6.21	MIPSel	239.2	128	87.22
ASUS RT-N56U	Router	2.6.21	MIPSel	249.3	128	78.56
Cisco Linksys E4200	Router	2.6.22	MIPSel	239.2	64	18.63
D-Link DCS-942L	IP Camera	2.6.28	ARM	534.5	128	38.75
SimpleHome XCS7-1001	IP Camera	3.0.8	ARM	218.7	32	1.90
Provision PT-737E	IP Camera	3.4.35	ARM	218.7	32	3.88

Table 1: IoT devices evaluated (CPU speed in BogoMIPS).

2.3 Requirements for an IoT Defense Solution

In this section, we specify desired requirements for a host-based IoT defense solution that is resistant against our attacker model. In particular, a defense solution should meet the following requirements:

(1) Real-Time Detection: Since we aim to prevent any unknown action on an IoT device, a defense solution must be capable of detecting any unknown program upon its execution.

(2) Lightweight Overhead: IoT devices are extremely resource constrained, and thus provide only limited processing and storage resources. Therefore, it is not possible to utilize conventional security approaches used in PC environments (e.g., machine learning or complex heuristics approaches). With this in mind, a defense solution should be conservative in terms of resource consumption and should only utilize existing dependencies.

(3) Tamper-Proof Protection: Since the attacker has superuser privileges, he may terminate or bypass a defense solution deployed on an IoT device. Therefore, a defense solution should be resilient against such a powerful attacker.

(4) Wide Coverage: It is important to protect a wide range of IoT devices (e.g., printers, IP cameras, Wi-Fi routers), taking into account that a significant portion of the existing IoT devices is already considered legacy and moreover may lack updates.

(5) Independence: The deployment of a defense solution must not be dependent on a manufacturer; both the user and a manufacturer must be capable of deploying it.

(6) Ease of Bootstrapping: With regard to the deployment of a defense solution mentioned above, we further argue that a defense solution should be capable of being deployed with small effort and should not require kernel recompilation of the IoT device's OS.

2.4 Design Problems and Options

We analyzed options for developing a defense solution and identified additional constraining factors that should be considered as well. Initially, we conjectured that the most straightforward option is to utilize features provided by Linux, such as *KProbe*[1] or *inotify*.[2]

[1] https://www.kernel.org/doc/Documentation/kprobes.txt
[2] http://man7.org/linux/man-pages/man7/inotify.7.html

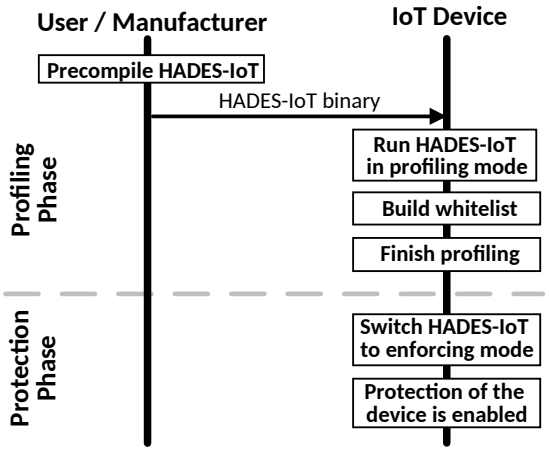


Figure 1: Bootstrapping of HADES-IoT.

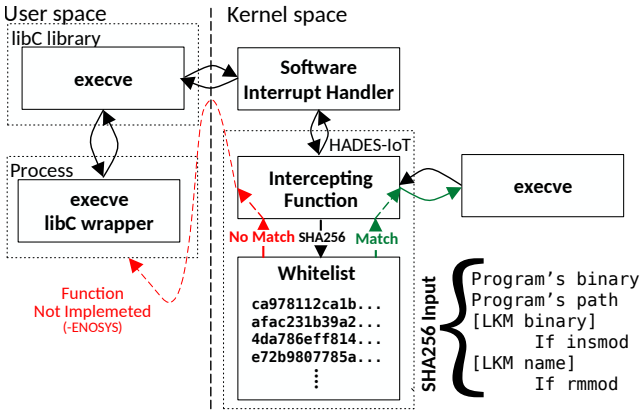


Figure 2: Flow of the *execve* execution with HADES-IoT.

However, after examination of several IoT devices (see Table 1) we found that these features are not supported in neither of them.

Another design aspect we considered is the lightweight complexity and low resource requirements of the defense solution. In particular, the most important resources for a defense system are CPU and memory. We measured the normal utilization of these resources by the IoT devices in this study and found out that while there is a reasonable reserve of CPU utilization, the CPU performance is often low. Therefore, we must ensure that the defense solution minimizes CPU consumption only to the extent needed, and the device's performance and availability is not affected. We also observed that there is only a small amount of free memory on IoT devices (e.g., $\leq$2MB), but not the whole free space can be utilized for a defense solution, as some applications might rely on it.

Since the challenge is to detect unknown processes in real-time upon their spawning, the Linux process scheduler is another limiting factor. In the user space environment, processes compete for the CPU, and the process scheduler makes decisions regarding the CPU and time allocations for the processes. Therefore, if a defense solution were implemented in user space, there is no guarantee that it would be "scheduled as running" when a new process is spawned; hence, malicious processes might be missed or detected too late.

System call interception is a suitable technique capable of addressing the issue of execution priority. With an appropriate set of intercepted system calls, this technique enables us to "catch" all new processes upon their spawning. In general, there are two options for performing system call interception. The first option is *libC library hooking*, which is performed in the user space, and the second option is the interception of system calls through a *loadable kernel module* (LKM) running in kernel space. Although libC hooking is easier to develop with more freedom as compared to creating an LKM, it is not suitable for our case because the IoT environment is diverse, and each manufacturer uses custom Linux that can be compiled with various (or custom) libraries/their versions (including libC). Attackers address these issues by compiling their malware statically and thus cover as many IoT devices as possible. This fact renders libC hooking unusable for the detection of the majority of malware. Hence, we identify LKM as the most feasible solution, which can fulfill most of the requirements defined in Section 2.3.

3 HADES-IOT

We propose a host-based anomaly detection system targeted for IoT devices, called HADES-IoT. Most of the requirements specified above have been fulfilled since we chose to adopt the LKM approach that utilizes the system call interception technique and, more specifically, intercepts the *execve* system call. Using the LKM approach, we are able to install HADES-IoT into a Linux kernel at any time; moreover, with this approach there is no need to recompile the kernel. The only requirement for ensuring that HADES-IoT can run on an IoT device is that HADES-IoT needs to be distributed in binaries that are precompiled (see details in [5]).

HADES-IoT is based on the whitelisting approach. The idea of this approach is that only programs that are known to run on an "uninfected" off-the-shelf device are allowed to run. In order to build a whitelist of benign programs, profiling must be performed once for each device. This may be viewed as impractical due to the possibility that some benign programs may be missed during profiling. Nevertheless, we deal with this issue and allow the whitelist to be updated at runtime (see Section 3.5).

3.1 Bootstrapping

We distinguish between two modes of HADES-IoT: 1) **a profiling mode** and 2) **an enforcing mode**. HADES-IoT is bootstrapped on a device in two stages (see Figure 1). First, HADES-IoT is precompiled and delivered to the device, and the kernel's initialization file is modified accordingly to ensure that HADES-IoT is always executed when the device is booted. Once executed, HADES-IoT enters the profiling mode, in which, it monitors and collects information about all calls to *execve* and keeps updating the whitelist. The profiling stage ends when no new process is detected during a specified time. We emphasize that during the profiling, also a restart of the device is performed, which enables to update the whitelist with all the programs executed at the boot time. In the last stage, HADES-IoT is switched to the enforcing mode, protecting the device using the whitelist.

3.2 Detection Process

The most important feature of the detection process is the intercepting function. The process of interception is depicted in Figure 2. Upon deployment, HADES-IoT locates the system call table and

saves the address of the *execve* system call found in the table. Next, *execve*'s address in the table is replaced by the address of the intercepting function. This ensures that each time the *execve* is called, the software interrupt handler calls the intercepting function instead of the original *execve* system call. Once the intercepting function is executed, it first reads the parameters passed to the *execve* system call (i.e., the path of the program to be executed). Next, the function computes a SHA256 digest out of the program's binary content, its path, and other data, depending on the particular circumstances (see Section 3.3). Using the computed digest, the intercepting function looks for a match on the whitelist of all authorized programs. If a match is found, the process is allowed to run, and therefore the intercepting function performs a call to the original *execve* system call. However, if a match was not found, the intercepting function returns "$-ENOSYS$" error code, which causes termination of the process, stopping the execution of any unauthorized action.

3.3 Whitelist Design

When HADES-IoT is successfully bootstrapped, each call to the *execve* system call is intercepted, followed by a search in the whitelist for a program that is requested to run. An inefficiently designed search process would cause the IoT device exhibit slow response time, particularly when the device has a large number of periodically spawned processes. For example, if the whitelist were naïvely designed as a linked list, the asymptotic time complexity of the search routine would be equal to $\Theta(n)$. This means that the larger the whitelist is, the longer the imposed delay. To cope with such a delay, we designed the whitelist as a hash table, which enables us to reach an asymptotic time complexity of $\Theta(1)$ for a search routine, thus providing constant delay for the search routine.

IDs in the Whitelist. Each item in the whitelist contains ID and represents a program authorized to run on a device. The ID of an item is a SHA256 digest computed from a program's binary, concatenated with the path to the program, and in certain circumstances, with other additional data. The reason for such a combination is to distinguish symbolic links from the executable they point to. An example of concerning executable that is heavily utilized in Linux-based IoT devices is *BusyBox* – it combines a set of common Unix utilities under a single executable, while particular utilities are accessible through symbolic links. Therefore, if we were to compute the digest out of just the binary content of the passed program, we would obtain the digest of BusyBox for all of the utilities. However, after adding a path element to the digest computation, the resulting digest is unique for each of the utilities.

Nevertheless, there are cases in which this approach is not sufficient. Hence, we have to handle such cases with more fine-grained whitelisting, in which, additional context dependent data must be added to the input for SHA256 (see Section 3.4.2 and 3.4.3).

3.4 Tamper-Proof Features of HADES-IoT

We assume that the attacker is provided with superuser privileges once the IoT device has been compromised. Therefore, we need to ensure that the attacker who is aware of the presence of HADES-IoT is unable to terminate or modify it. In the following, we describe protection mechanisms against possible attacks that focus on tampering with HADES-IoT.

3.4.1 ***Modification of HADES-IoT/Initialization File.*** To prevent a manipulation of HADES-IoT's binary and the *init* file by an attacker, HADES-IoT loads its binary and the *init* into the protected memory on boot and when a restart of a device is requested, HADES-IoT's binary as well as the *init* are (re-)written to the storage, regardless of whether the original version was modified or not. Hence, any malicious modification, removal, or rename of the original files is prevented.

3.4.2 ***Loading Malicious LKMs.*** As a protection mechanism, each execution of *isnmod* must be verified against the allowed and known executions, requiring more fine-grained indexing to the whitelist. In contrast to the whitelist indexing of standard binaries, in the case of *insmod* the index to the whitelist is computed as a cryptographic hash from: 1) binary content of *insmod*, 2) its path, and 3) the binary content of an LKM that is requested to load to a kernel. This ensures that only known LKMs from the profiling stage are allowed to load again.

3.4.3 ***HADES-IoT Uninstallation.*** The prevention of this attack is almost the same as in the previous case. The only difference is that instead of adding the binary content of the LKM to the hash computation, it is sufficient to use the name of the LKM, as the name unambiguously identifies a kernel module that has been already installed.

3.4.4 ***Memory Tampering.*** To prevent memory tampering, HADES-IoT takes advantage of the fact that it is integrated into the kernel's memory. Therefore, user space programs cannot reach the kernel, since accessing the kernel space is forbidden and results in a segmentation fault. The only chance for an attacker to tamper with HADES-IoT's memory is to get into the kernel space as well. Countermeasures that prevent the attacker from loading anything into the kernel are presented in Section 3.4.2.

3.5 Extensions

We introduce a user-space application running on an IoT device, which informs the owner about alerts, enables him to extend the whitelist, and finally allows him to perform updates of an IoT device. Since IoT devices rarely provide dependencies for asymmetric cryptography, the application uses authentication with Merkle signatures, requiring only a cryptographically secure hash function, while providing resilience against quantum computing. Next, we add interception of kill system call, since it can be invoked by a kill utility or a dedicated shell built-in function (see details in [5]).

4 EVALUATION

We developed a proof-of-concept implementation of HADES-IoT and tested it on seven IoT devices (see Table 1). First, we experimented with profiling time and determined the minimal amount of time required to extract an accurate profile. Then, we evaluated the detection performance of HADES-IoT on vulnerabilities exploited by recent IoT malware, and finally we measured resource consumption.

4.1 Profiling Period

During bootstrapping, HADES-IoT must run in the profiling mode to extract the profile of an IoT device. The longer HADES-IoT runs in

IoT Device	Executables Found	Whitelist size / profiling time 1h	2h	4h	1h (user inter.)
Netgear WNR2000v3	526	12	12	12	61
ASUS RT-N16	638	5	5	5	38
ASUS RT-N56U	375	3	3	3	6
Cisco Linksys E4200	399	9	9	9	11
D-Link DCS-942L	1256	20	20	20	105
SimpleHome XCS7-1001	588	4	4	4	29
ProVision PT-737E	482	5	5	5	9

Table 2: Comparison of profiling time vs. whitelist size.

this mode, the more accurate profile is extracted. However, it can be inconvenient for a user when the profiling takes too long. Therefore, we conducted an experiment in which we determined the boundaries on the amount of profiling time needed to obtain an accurate profile (see Table 2). We can see that after one hour of profiling no new processes were found on any of the devices, which means that an accurate profile can be obtained even after one hour of profiling. On the other hand, there is the possibility that a new program might be executed after a four hour profiling period (e.g., a scheduled job). However, with the extension described in [5], any missing program can be added to the whitelist after the profiling phase.

Next, we measured the difference in the whitelist size when a user interacts with the GUI of a device and cases in which there is no user interaction. The results show that on some devices, such as *Netgear WNR2000v3* and *D-Link DCS-942L*, the whitelist increases significantly, while on devices, such as *Cisco Linksys E4200* and *ASUS RT-N56U*, the increase is only small. These results suggest that it is important to interact with the device during the profiling period, otherwise many programs could be missed, leading to a less accurate profile.

Finally, we compared the number of all executables to the number of executables presented in the whitelist. Table 2 shows that each device contains a large number of executables that are never used. If an attacker were to compromise the device, none of these executables would be used due to the protection provided by HADES-IoT; the attacker is strictly limited to the executables in the whitelist.

4.2 Effectiveness of Detection & Prevention

To demonstrate the prevention capabilities of HADES-IoT, we performed several attacks that exploit vulnerabilities used by recent real-world IoT malware. We describe these attacks in the following.

4.2.1 ***Enabled Telnet with Default Credentials & Mirai.*** The devices with enabled Telnet service by default (e.g., SimpleHome IP camera) are potentially vulnerable to Mirai. However, with HADES-IoT even such a default misconfiguration does not cause harm, since execution of any unauthorized binary is terminated upon its spawning, as witnessed by our evaluation.

4.2.2 ***[CVE-2017-8225] & IoTReaper, Persirai.*** We bootstrapped HADES-IoT on the vulnerable IP camera (i.e., ProVision PT-737E) and executed the exploit. In the first step of the exploit, the credentials are retrieved by reading the *system.ini* file. Since this is handled by the HTTP server that is in the whitelist, HADES-IoT allows this action. In the second step, the remote command is sent through FTP configuration CGI. According to HADES-IoT's logs, this executes the *chmod* utility, as well as *ftpupload.sh* that executes the command. However, since none of these executables are in the whitelist, HADES-IoT terminates both of them upon their execution and stops the attack.

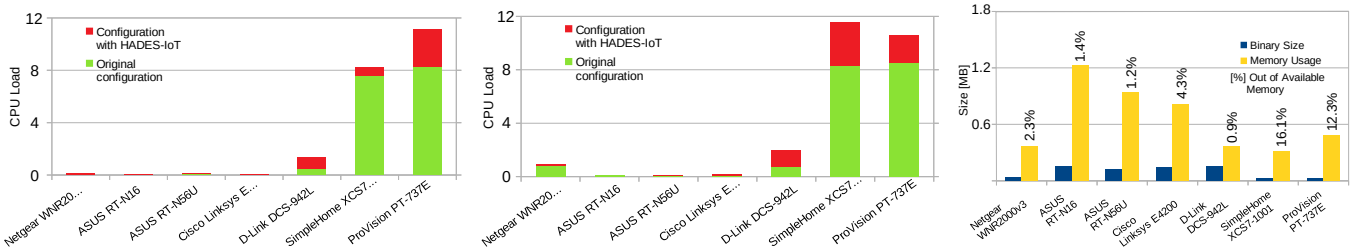
4.2.3 ***[CVE-2014-9583] & VPNFilter.*** We tested exploitation of this vulnerability on the ASUS RT-N56U device protected by HADES-IoT in two scenarios: with disabled and enabled Telnet. With Telnet disabled (i.e., default option), an attempt to compromise the device is detected upon exploit execution. However, in the case in which Telnet is enabled, the attack is detected in its later stage – upon malware download or its execution, depending whether an HTTP client (e.g., wget, curl) is in the whitelist (depending on user's profile).

4.2.4 ***TelnetEnable Magic Packet & VPNFilter.*** We tested the exploitation of this vulnerability on the Netgear WNR2000v3 device. Like the previous vulnerability, if the Telnet service is disabled, the attack by VPNFilter is detected by HADES-IoT when it begins. With enabled Telnet, the attack is thwarted as soon as any unauthorized process is spawned (i.e., execution of the malware at the latest).

4.3 CPU and Memory Overhead

An important aspect of a host-based defense system for IoT devices is low performance overhead. We conducted an experiment in which we measured the CPU utilization caused by HADES-IoT (see Figure 3a and 3b) and the memory demands (see Figure 3c). More specifically, we measured an average CPU load that represents how much work has been done on the system in the previous *N* minutes [7] – in our case we used five minutes. Although this metric has no clear boundaries, it is the most available option on all IoT devices; common utilities such as *top* do not show LKMs and their resource utilization, hence this information cannot be obtained from them. Figure 3a shows that when a device is idle, there is usually only a small amount of overhead. Note that in some devices (e.g., *ProVision PT-737E*) the CPU load is significantly higher than on other devices. This is caused by the periodic execution of hundreds processes that, for example, extract various information from the device itself. Since HADES-IoT checks every executed process, the overhead imposed is higher compared to less active devices – the imposed overhead comprises 34.6% and 36.9% of the total overhead for the idle case and the case of user interaction, respectively. Furthermore, we can see in Figure 3b that on some devices (e.g., *SimpleHome* and *D-Link* devices), HADES-IoT imposes greater overhead when a user interacts with the device. This is caused by spawning additional processes to handle the request, while on other devices the request can be handled by already running processes.

Next, we measured memory and storage demands of HADES-IoT, and the results (presented in Figure 3c) demonstrate low memory demands in contrast to available space: 5.5% on average. Regardless of the same source code, the binary size and average memory usage varies for each device. One of the reasons is that various cross-compilers were used to compile HADES-IoT for each device. However, what does have the highest impact on the binary size is the configuration of the Linux kernel against which HADES-IoT is compiled. For example, debug symbols are included in the HADES-IoT's binary due to the kernel configuration, which increases the size of the binary. Also, note that differences in memory usage across



(a) CPU load (idle) (b) CPU load (with user interaction) (c) Memory and storage utilization

Figure 3: Utilization of resources by HADES-IoT.

various devices are caused by the introduced tamper-proof mechanisms – the integrity of HADES-IoT's binary and *init* file that must be loaded into the memory of HADES-IoT. In detail, these files have a different size on each device, hence loading them into the memory causes different memory consumption for each device.

5 RELATED WORK

In this section, we discuss research aimed at host-based intrusion detection systems for IoT devices. An anomaly-based approach is provided by Yoon et al. [20]; the authors present a lightweight method based on the distribution of system call frequencies. However, the authors only consider attacks that alter system calls in benign programs and use only one sample for their evaluation (i.e., `Motion`). A lightweight IDS that focuses on smart meters is proposed in [16]; this research is based on system call sequences, where the benign program is represented by a finite state machine (FSM). The system calls are captured by the *strace* utility and stored in a log file. The second component compares the captured system calls stored in the log with the FSM once per 10 seconds to ensure low demands on resources. However, the presented IDS is designed to be used on smart meters only, so it is trained on a single executable. On top of that, it requires the addition of annotations to the smart meter software by its developers. The work of Agarwal et al. [1] presents a concept for anomaly detection that uses context-sensitive features based on Ball-Larus path profiling. However, this approach requires the source code be instrumented, which facilitates the recording of function calls during execution. An et al. [2] propose behavioral anomaly detection aimed at home routers. This research employs three semi-supervised algorithms (i.e., principal component analysis, one-class SVM, and a naïve detector based on unseen n-grams) utilizing captured kernel-level system calls to determine whether a device has been compromised. Their results show that all three algorithms employed achieved a 100% detection rate with a low false alarm rate, but the overhead inflicted by the approach is not evaluated. The downside of the approach is that full kernel recompilation with enabled *ftrace* support is required. Su et al. [15] present a lightweight image recognition technique for malware classification based on convolutional neural network. The authors achieved 94% accuracy and 81% accuracy for two-class and three-class classification, respectively. However, the authors admit that their approach is susceptible to complex code obfuscation.

6 CONCLUSION

In this paper, we proposed HADES-IoT, a host-based anomaly detection system for IoT devices, which provides proactive detection and tamper-proof resistance. HADES-IoT is based on a whitelist that is build during the profiling, and it utilizes system call interception technique realized within the loadable kernel module (LKM) of Linux-based OSes. Leveraging the LKM, HADES-IoT gains complete control of the execution of all user space programs, and any execution of an unauthorized binary can be aborted upon its start. HADES-IoT is lightweight in terms of its size, memory, and CPU demands. Computational overhead is only influenced by the number of spawned processes on the device, but not by operations with the whitelist – searching in the whitelist has a constant time complexity. In the evaluation, we showed that extraction of an accurate device profile can be performed in an hour, and we demonstrated 100% effectiveness in the detection of four kinds of attacks.

REFERENCES

[1] A. Agarwal et al. Detecting abnormalities in IoT program executions through control-flow-based features. In *Internet-of-Things Design and Implementation (IoTDI)*, pages 339–340. IEEE/ACM, 2017.
[2] N. An et al. Behavioral anomaly detection of malware on home routers. In *Malicious and Unwanted Software (MALWARE)*, pages 47–54. IEEE, 2017.
[3] M. Antonakakis et al. Understanding the Mirai botnet. In *USENIX Security Symposium*, pages 1092–1110, 2017.
[4] M. Boltz et al. New Methods and Combinatorics for Bypassing Intrusion Prevention Technologies. Technical report, Stonesoft, 2010.
[5] D. Breitenbacher et al. HADES-IoT: A Practical Host-Based Anomaly Detection System for IoT Devices (Extended Version). *preprint arXiv:1905.01027*, 2019.
[6] P. Fogla et al. Polymorphic Blending Attacks. In *USENIX Security Symposium*, pages 241–256, 2006.
[7] N. J. Gunther. Unix load average, part 1:, 2010. https://www.teamquest.com/import/pdfs/whitepaper/ldavg1.pdf.
[8] I. Homoliak. *Intrusion Detection in Network Traffic*. PhD thesis, Dissertation, Faculty of Information Technology, University of Technology Brno, 2016.
[9] I. Homoliak et al. Improving network intrusion detection classifiers by non-payload-based exploit-independent obfuscations: An adversarial approach. *EAI Endorsed Transactions on Security and Safety*, 5(17), 12 2018.
[10] G. V. Hulme. Embedded system security much more dangerous, costly than traditional software vulnerabilities, 2012.
[11] K. L. Lueth. State of the IoT 2018: Number of IoT devices now at 7b – market accelerating, 2018.
[12] R. v. d. Meulen. Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016, 2017.
[13] R. Newell. The biggest security threats facing embedded designers, 2016.
[14] D. N. Serpanos and A. G. Voyiatzis. Security challenges in embedded systems. *ACM Trans. Embed. Comput. Syst.*, 12(1s):66:1–66:10, Mar. 2013.
[15] J. Su et al. Lightweight classification of IoT malware based on image recognition. In *Annual Computer Software and Applications Conference (COMPSAC)*, pages 664–669. IEEE, 2018.
[16] F. M. Tabrizi and K. Pattabiraman. A model-based intrusion detection system for smart meters. In *International Symposium on High-Assurance Systems Engineering (HASE)*, pages 17–24. IEEE, 2014.
[17] O. I. S. Team. Owasp IoT top 10, 2018.
[18] TrendMicro. VPNFilter-affected devices still riddled with 19 vulnerabilities, 2018.
[19] G. Vigna et al. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Computer and Communications Security (CCS)*, pages 21–30. ACM, 2004.
[20] M.-K. Yoon et al. Learning execution contexts from system call distribution for anomaly detection in smart embedded system. In *Internet-of-Things Design and Implementation (IoTDI)*, pages 191–196. IEEE/ACM, 2017.
[21] J. young Cho. Linux takes lead in IoT market keeping 80% market share, 2017.