# MIDAS: Safeguarding IoT Devices Against Malware via Real-Time Behavior Auditing

Yiwen Xu†, Zijing Yin†, Yiwei Hou, Jianzhong Liu, Yu Jiang*
*Tsinghua University, Beijing, China*

***Abstract*—The number of IoT devices on the Internet has surged recently, accompanied by a barrage of large-scale IoT malware infections breakouts. Designing security mechanisms for IoT devices poses significant challenges due to constantly changing malware variants that have numerous camouflage strategies, limited hardware resources and heterogeneous architectures. In this paper, we propose MIDAS, an adaptive safeguard framework for Linux-based IoT devices to defend against malwares with the real-time behavior auditing mechanism. First, we construct a stable and abstract behavior paradigm through behavioral characteristic extraction of 115,970 malwares. Then, based on the behavior paradigm, MIDAS can 1) monitor suspicious behaviors of break-in programs in real-time driven by our built-in SELinux policy customized for malware defense, 2) aggregate behaviors of the program's submodules with homology tracing and 3) summarize these behaviors into abstract behavior pairs to unveil a possible IoT malware. Using the aforementioned real-time behavior auditing, MIDAS can constrain mutating and camouflaged malwares to protect discrepant IoT devices from being compromised while maintaining low overheads.**

**We thoroughly evaluated the defense capabilities of MIDAS. On the benchmark dataset, MIDAS successfully constrained up to 94.46%, 91.79% and 88.34% of 115,970 malware samples on ARM, MIPS and MIPSEL architectures, with less than 1.8MiB of memory consumption and 0.54% CPU usage. Furthermore, we deployed virtual IoT devices worldwide to examine the performance of MIDAS when defending against real-world attacks. Over a duration of 25 days, these devices suffered from 971,951 attacks originating from 71,979 intruding malwares and 48,805 unique IPs distributed in 167 countries. For devices with MIDAS protection, the number of compromised incidents decreases by $343.1\times$, and the duration of continuous operation is $179.2\times$ greater than devices without MIDAS on average. The evaluation results demonstrate that MIDAS can effectively safeguard IoT devices with minimal resource consumption.**

***Index Terms*—IoT Malware, Embedded Firmware, Adaptive Safeguard, Behavior Auditing.**

## I. INTRODUCTION

Internet of Things devices, or IoT devices, refer to billions of Internet-connected devices. Many of these devices run Linux [1], and are power-limited, cost-constrained, and heterogeneous without any protection against IoT malwares [2]. The typical lifecycle of IoT malware is composed of four parts: Infection, Reconnaissance, Persistence and Impacts. After infection from common entry points, IoT malwares first gather device's information for reconnaissance. Then they will maintain their foothold on the device for persistence and further achieve impacts for malicious goals. Nowadays, numerous IoT malware families like Mirai support multiple architectures, iterate variants constantly and infect more than 10K devices a day [3]. Such a rising security threat illustrates the necessity of a safeguard mechanism for IoT devices. Recent studies have also concluded that immediate security efforts are required to defend against IoT malware breakouts [4].

In recent research, malware mitigation frameworks based on static and dynamic features have achieved good performance in traditional domains. Among them, static signature-based approaches [5] [6] extract patterns like bytecode or API calls from malware samples. Yet, these approaches are susceptible to evasion techniques, like obfuscation [7], and are constrained by the limited IoT device's storage for patterns. In contrast, current dynamic analysis methods [8] [9] can be more accurate by identifying the malware via runtime information. However, the required hardware resources are too heavy for IoT devices.

To safeguard IoT devices against malwares effectively, we encounter the following *challenges*. First, IoT malwares mutate rapidly based on their original binaries. Without analyzing the programs' behavioral semantics, patterns developed for specific samples, such as YARA, can be easily bypassed by subsequent variants with obfuscation or code polymorphism. Thus, how to extract typical and stable behavioral characteristics of IoT malwares poses significant challenges.

Second, IoT malwares often apply various camouflage methods like self-replication to decentralize malicious intentions into different submodules. Insufficient malware tracing makes existing security methods difficult to provide a complete view of malware' behaviors throughout their lifecycle, reducing the protection's effectiveness. How to comprehensively analyze homologous relations under various camouflage strategies of the intruding malware is an obstacle to tackle.

Finally, most IoT devices have constrained hardware resources and heterogeneous architectures. Current on-device safeguard approaches require considerable storage or computational resources for real-time analysis, which exhibits heavy overheads. Meanwhile, existing approaches lack support for heterogeneous devices and are highly coupled with the target architectures. How to implement a lightweight safeguard

*Yu Jiang is the corresponding author.
†Yiwen Xu and Zijing Yin contributed equally to this research.
Y. Xu, Z. Yin, Y. Hou, Jianzhong Liu and Y. Jiang are with the School of Software, Tsinghua University, Beijing 100084, China (e-mail: xuywen14@gmail.com, aurora@europe.com, houyiweiyw@gmail.com, liujz21@mails.tsinghua.edu.cn, jiangyu198964@126.com).

Manuscript received April 07, 2022; revised June 11, 2022; accepted July 05, 2022. This article was presented at the International Conference on Embedded Software (EMSOFT) 2022 and appeared as part of the ESWEEK-TCAD special issue.

This research is sponsored in part by the NSFC Program (No. 62022046, 92167101, U1911401, 62021002, 62192730), National Key Research and Development Project (No. 2019YFB1706203, 2021QY0604) and MIIT Project-(Design of intelligent networked vehicle based on SOA central control.

framework compatible with devices of different architectures needs to be adequately addressed.

To address these challenges, we propose MIDAS, an adaptive safeguard framework to defend Linux-based IoT devices against malwares. We systematically analyze 115,970 IoT malwares over 28 families to acquire their sensitive behaviors. IoT malwares' behaviors access various system resources that can be categorized by their security sensitivity. Security sensitivity indicates the level of permissions to access some given resource, i.e., the operations allowed for certain programs to perform on the given resource. Resources with similar security sensitivities generally have similar functionalities. This allows us to abstract behaviors of malwares through using security-sensitivity-based resource categories (represented as security context), as the security sensitivity of the accessed resources of a malware's given intentions remain relatively constant and can be carried across variants. Thus, malwares' behaviors can be abstracted into pairs consisting of security contexts and behavior classes, which are then mapped to stages of typical IoT malware lifecycle. Such a *behavioral characteristic extraction* process can construct a robust and stable behavior paradigm applicable to mutating IoT malwares.

The paradigm portrays the suspicious behaviors throughout the entire lifecycle of IoT malwares. With this behavior paradigm, we can perform *real-time behavior auditing* on break-in programs to safeguard IoT devices. Firstly, based on a built-in customized malware defense policy, suspicious behaviors of the intruding program can be recorded as AVC messages by the SELinux kernel module with low runtime costs, allowing MIDAS to automatically monitor these operations in real-time cheaply. Such an architecture-agnostic procedure also makes MIDAS compatible with heterogeneous IoT devices. Secondly, homology tracing is designed to aggregate operations scattered among multiple submodules of the break-in program to fully depict its activities with various disguises. In this way, operations derived from homologous processes of the program will be traced as a behavior message sequence and summarized into abstract behavior pairs. Finally, if MIDAS can assemble abstract pairs to unveil a complete IoT malware lifecycle, the malicious attempts will be blocked and the break-in program will be constrained from violating the system.

We implemented and evaluated the performance of MIDAS on both multi-source benchmark malwares and real-world attack scenarios. Specifically, on the benchmark dataset, MIDAS successfully withstood and constrained up to 94.46%, 91.79%, 88.34% malwares on ARM, MIPS, and MIPSEL architectures respectively, with less than 1.8MiB of memory and 0.54% CPU. Furthermore, we deployed virtual IoT devices with different architectures to public networks in ten countries to examine the performance of MIDAS against current real-world attacks. Half of the devices were protected with MIDAS, while the other half were not. Suffering from a total of 71,979 intruding malwares and 971,951 attacks (i.e. login sessions) originating from 167 countries, devices without MIDAS became compromised 11,323 times in total, while devices safeguarded by MIDAS were compromised 33 times only due to connectivity loss by external DDoS attacks. In summary, with MIDAS, the number of compromised incidents decreases by $343.1\times$ and the duration of continuous operation increases by $179.2\times$ than devices without MIDAS. This fully demonstrates that MIDAS can effectively protect IoT devices with heterogeneous architectures at low run-time costs.

Our contributions are summarized as follows:

- We propose an adaptive and efficient safeguard framework for IoT devices based on real-time behavior auditing mechanism. Such a methodology can be applied to heterogeneous IoT devices to defend against mutating malwares under various camouflages with low overheads.
- After constructing the behavior paradigm by the behavioral characteristic extraction of 115,970 IoT malwares, we implement MIDAS[1] to monitor suspicious operations, trace homologous processes, and summarize the abstract behaviors of break-in programs to identify possible malwares based on the extracted paradigm.
- We evaluated MIDAS on benchmark and real-world attack scenarios. Results show that MIDAS defended up to 94.46% of benchmark samples, with less than 1.8MiB of memory and 0.54% CPU usage. Meanwhile, in 25-day online experiments, devices suffered from 971,951 attacks, and MIDAS-protected devices are $343.1\times$ less to be compromised than ones without MIDAS.

## II. BACKGROUND

SELinux is an access control module in the Linux kernel. It supports fine-grained behavior auditing. It is an implementation of Flux Advanced Security Kernel (FLASK), a methodology that uses loadable policy to achieve flexible security protection. The policy loaded in SELinux is called SELinux Policy. It specifies what kind of behaviors or resource accesses need to be audited.

A typical policy consists of three aspects: *labeling*, *transition* and *rules*. (1) *Labeling* indicates system resource classification. Each system resource, including files, networks, processes, system capabilities, etc., can be categorized based on its functionalities. Resources with the same access permissions will be labeled with the same security contexts. With the *labelling* procedure, system resources originally embedded in the firmware are allocated with security contexts. (2) The security contexts of newly generated resources (e.g., output files of programs), on the other hand, are determined based on *transition* in the policy, implemented with *type transition* statements. They can specify what security context will be labeled to new resources generated from existing programs. Through these two mechanisms, the security context of each system resource, including both existing resources and newly generated resources can be explicitly assigned. The resources with the same security context will be applied with the same rules. (3) Each *rule* in the policy can designate a set of behaviors that need to be audited. We focus on three items when designing a rule: security context of the resource requester (i.e., source security context, referred to as *scontext*), security context of the requested resource (i.e, target security context, referred to as *tcontext*) and the type of the operation

[1]The prototype of MIDAS, the malware dataset and supplementary materials are available at https://anonymous.4open.science/r/MidasDefense-6F7C

(e.g, read, write, etc., referred to as *behavior class*). Each rule can trigger the SELinux module to audit the behaviors of the *behavior class*, performed by processes with *scontext* to resources with *tcontext*.

```
{ add_name } for pid=1871 comm="cat" name="S99jcfk"
    scontext=u:r: suspicious.subj tcontext=u:r: file.initscriptfile tclass=dir
```

Listing 1: Example of an AVC message. This message is generated when the break-in process of HideNSeek malware, labeled with *suspicious.subj*, duplicates itself as *S99jcfk* to */etc/rc.d* for persistence.

SELinux module utilizes Access Vector Cache (AVC) that automatically cache recent audit decisions of operations to reduce overhead. The audited behaviors will be recorded as AVC messages. Each message contains *behavior class*, *scontext*, *tcontext* and other information. As shown in Listing 1 with highlight items, this AVC message records an *add_name* behavior (file creation in a directory) performed by *suspicious.subj* type of process to *file.initscriptfile* type of file resource (i.e., system initialization directories like */etc/rc.d*).

## III. MIDAS DESIGN

In this section, we introduce MIDAS' methodology, which consists of behavioral characteristic extraction and real-time behavior auditing. The overall process is shown in Figure 1.

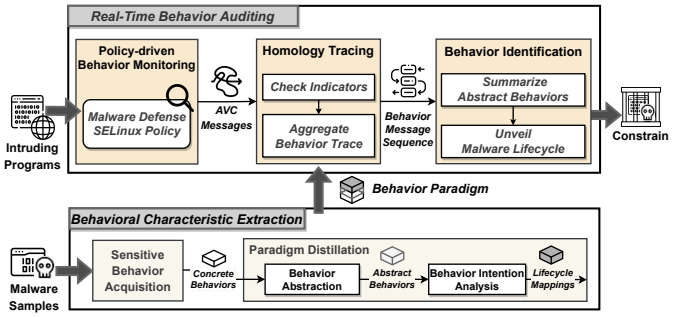


Fig. 1: MIDAS Overview. With Behavioral Characteristic Extraction, malwares' operations can be analyzed and abstracted to form a typical behavior paradigm. Utilizing this paradigm, Real-Time Behavior Auditing monitors intruding programs' suspicious behaviors, aggregate the behaviors conducted by homologous processes to a behavior message sequence and summarize them into abstract behavior pairs to unveil possible malware lifecycle and constrain malicious programs.

To construct the behavior paradigm of malwares, we use a behavioral characteristic extraction process. Through analyzing 115,970 samples, we acquire a series of sensitive behaviors conducted by IoT malwares. We abstract these concrete behaviors into pairs of security contexts and behavior classes. Then the intentions of these behaviors will be analyzed and further mapped to the IoT malware lifecycle. Such hierarchical abstraction can distill a general and robust paradigm that represents the correspondence from concrete operations to abstract behavior pairs and then the stages of IoT malware lifecycle, applicable to even the lastest variants.

Based on this paradigm, we design a real-time behavior auditing mechanism to defend against IoT malwares. Specifically, driven by the SELinux policy customized for malware defense, MIDAS monitors break-in programs' behaviors as AVC messages in real-time with low runtime costs. Then, MIDAS uses homology tracing to aggregate malicious behaviors of the camouflaged malware's submodules to form a behavior message sequence. Based on the obtained sequence, MIDAS summarizes operations into abstract behavior pairs. Finally, by assembling the pairs into stages of malware lifecycle according to the mappings in the paradigm, MIDAS can identify malicious programs and constrain them from violating the system.

### A. Behavioral Characteristic Extraction

We now illustrate the behavioral characteristic extraction procedure. It acquires sensitive concrete behaviors of IoT malwares and converts them into abstract behaviors which are then mapped to the malware lifecycle, constructing a stable behavior paradigm. Such a paradigm contributes to a more robust defense against variants in the further behavior auditing.

**Data Source.** We gather a total of 115,970 IoT malware samples from multiple sources to ensure diversity. The first part of the samples is collected from VirusTotal [10], which represents variants currently under study for its wide use by researchers. The second part comes from malwares captured by online honeypots deployed for up to three years [11], showing recently widespread variants. The remaining part contains samples of various underground repositories [12] [13] [4]. They have not received sufficient attentions from the academic community, which further increases the generalizability of our datasets. We de-duplicate all samples based on hash values, and then label each one with VirusTotal and AVClass [14].

**Sensitive Behavior Acquisition.** To acquire concrete behaviors of IoT malwares, we refer to the surveys of Cozzi et al. [15], Alrawi et al. [4] and Rieck et al. [16], which have been proven to be systematic and effective. We tailor these existing malware analysis methods for our sensitive behavior acquisition, especially the static, dynamic and capability differential analysis, to collect concrete behaviors of IoT malwares in our dataset. Based on these methods after customization, we can gather IoT malwares' behaviors and represent them as behavior pairs *<characteristic resource, operation type>*. In each pair, the first item represents the accessed resource of the behavior, and the second item represents the detailed type of the access to the resource indicated by system call trace. The analytical techniques here are not presented as our contributions. Analysis details and the dataset can be referred to the aforementioned supplementary material[1].

**Paradigm Distillation.** Based on the aforementioned analysis methods, we acquire concrete behavior characteristics of IoT malwares. Nevertheless, IoT malwares usually evolve rapidly based on their original versions, which leads to constant changes in specific resources they accessed. For effective defense, we should focus not only on the concrete operations of collected samples but also other functionally related behaviors targeted at resources with similar security sensitivity to achieve the same intention. Therefore, we apply an abstraction methodology to distill a typical malware behavior paradigm as

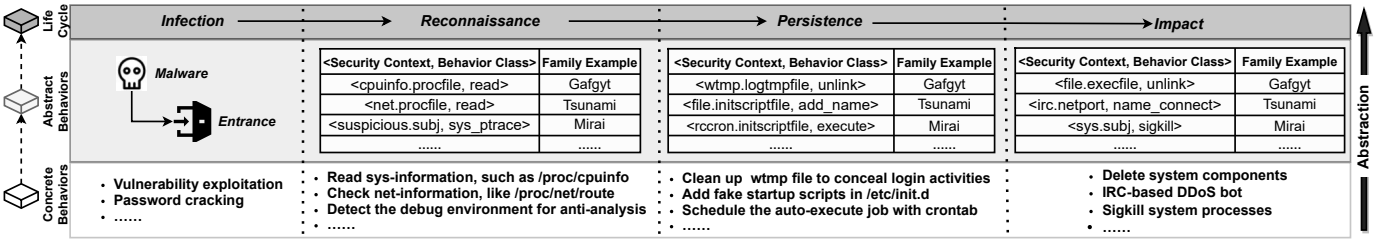


Fig. 2: The behavior paradigm of IoT malware. The bottom layer represents concrete behaviors of samples, while the middle layer is the corresponding abstract behavior pairs, and the top layer describes stages of the malware lifecycle.

shown in Figure 2, which can correspond concrete operations to abstract behavior pairs by behavior abstraction, and map them to the stages of malware lifecycle by behavior intention analysis. This hierarchically abstracted paradigm can make MIDAS more robust when confronted with new variants.

*1) Behavior Abstraction:* First, we abstract concrete behavior pairs *<characteristic resource, operation type>* into pairs of target security contexts and behavior class, i.e. *<security context, behavior class>*. Security contexts support the classification of resources, such as files, processes, network ports. Based on the standard SELinux policy in IoT domain, DSSP [17], we can obtain and customize fine-grained security contexts of each characteristic resource used in the behavior. The behavior class item represents operation categories of the access to the resource, including common types such as read, write, etc., as well as sensitive system capabilities like *sys_ptrace* (ptrace to others) or *sys_nice* (change priority). Overall, the pair *<security context, behavior class>* can be regarded as an abstraction of a certain kind of behaviors from a security sensitivity perspective. Even if concrete behaviors to achieve similar malicious intentions might change in new variants, their operations' security contexts and behavior classes, i.e abstract pairs, remain relatively consistent and stable.

Take behaviors in the "Persistence" stage of the malware lifecycle as an example shown in Figure 3. To auto-start after reboot on the device, malware families like HideNSeek copy itself to the */etc/rc.d* directory, randomly named as *S99lbdj* or *S99jcfk*. If we directly use this concrete path or filename as behavioral characteristics, it can be easily bypassed by new variants, like Mozi, that may accomplish the same persistent intention by adding a startup script named *S95baby.sh* to the */etc/init.d* directory. Nevertheless, the security context of these two accessed resources is the same, i.e. *file.initscriptfile* for their similar functionality, i.e. auto startup. Therefore, if abstract behaviors, i.e. the identical one *<file.initscriptfile, add_name>*, instead of concrete ones are utilized as representation in the paradigm, distinct persistence behaviors can be recognized. Even the two variants from different malware families (HideNSeek vs. Mozi) try to persist on devices with different file formats (binary vs. script), camouflaged as different names (*S99jcfk* vs. *S95baby.sh*), and located at distinct directories (*/etc/rc.d* vs. */etc/init.d*), the abstract paradigm can still be robust to identify behaviors for defense.

*2) Behavior Intention Analysis:* Second, we analyze each abstract behavior pair to figure out its malicious intention and build the mappings of abstract behaviors and malware lifecycle stages. Each abstract pair represents the malicious behaviors targeted at a particular class of resources, which achieves a certain intention to complete a part of the lifecycle. For each pair after de-duplication, we filtered out malware samples with such identical abstract behavior pair. Then according to the VirusTotal reports, samples with the longest prevalence, the earliest or the latest discovery time are picked out. They provide a good representation of how the concrete behavior corresponding to an abstract behavior pair changes throughout the mutation of malwares. We further send these samples to Joe Sandbox [18] for analysis. Based on the generated MITRE ATT&CK Matrix [19] [20], we can acquire the intention behind the behavior, which belongs to the corresponding stage of the malware lifecycle. For few behaviors that fail to be discovered by sandbox analysis, we manually analyze the behavior by reverse engineering the binary and estimate the stage of the malware lifecycle when the behavior is conducted.

In this way, we can determine which stage of the malware lifecycle this abstract behavior pair attempt to accomplish for the malicious intention and thus build mappings between pairs and stages as shown in the middle layer to the top layer of the paradigm in Figure 2. Such a process allows us to represent behaviors conducted along the whole malware lifecycle in a structured way with deeper abstraction, which further provides us with the ground truth of defending IoT malwares.

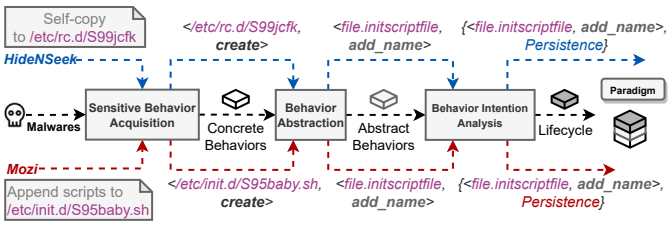


Fig. 3: The example workflow for distilling a set of behaviors to the paradigm. The data marked near the dashed line indicates the intermediate output after each procedure.

**Paradigm Overview** As shown in Figure 2, after the aforementioned procedure, we construct a hierarchical paradigm for malware behaviors represented with abstract and stable characteristics. First, we collect concrete behaviors based on tailored malware analysis methods in *Sensitive Behavior Acquisition*. For example, as illustrated in Figure 3, this procedure can obtain the concrete behavior pairs of HideNSeek and Mozi malwares as *</etc/rc.d/S99jcfk,create>* and *</etc/init.d/S95baby.sh,create>*. Then the concrete behaviors targeting the same resource category can be characterized into the same abstract behavior pair by *Behavior Abstraction*. That is, in Figure 3, distinct concrete behaviors of the two malwares can be abstracted to the same pair *<file.initscriptfile, add_name>*. Further, *Behavior Intention Analysis* obtains ma-

licious intentions behind behaviors to map abstract behaviors to malware lifcycle stages. Specifically, both behaviors of the two malwares in Figure 3 aims to achieve auto startup, hence being mapped to the persistence stage. Altogether, the conversion from concrete behaviors to abstract behaviors and then malware lifecycle allows MIDAS to characterize IoT malwares' behaviors in a robust way.

### B. Real-Time Behavior Auditing

Based on the constructed behavior paradigm, we further design a real-time behavior auditing mechanism to defend against malwares. Specifically, to monitor suspicious behaviors of break-in programs, we devise an SELinux policy for malware defense, named MD policy, based on the paradigm. It can trigger SELinux to record sensitive concrete behaviors of break-in programs as AVC messages in real-time with low overheads. Then we aggregate AVC messages of homologous processes produced by the same break-in entity to form a behavior message sequence during Homology Tracing. Next in Behavior Identification, aggregated AVC messages in the sequence can be summarized into abstract behavior pairs. By piecing up those pairs into stages of the malware lifecycle based on mappings in the paradigm, we can identify the maliciousness of break-in programs and further constrain them.

**Policy-driven Behavior Monitoring.** Implementing monitoring based on the built-in SELinux policy is an efficient and practical solution for heterogeneous Linux-based IoT devices. Linux is the top OS of IoT devices [1], and SELinux is a kernel module in it. This module can be ill-suited to be used on traditional domains like PCs, due to their changing functionalities and user interaction which requires policy adjustments. But most embedded devices are fixed-function and single-purpose, improving the SELinux usability in IoT domain. Thus with suitable SELinux policy loaded into the firmware, we can efficiently monitor sensitive behaviors of break-in programs in real-time as AVC messages, as shown in Figure 4.

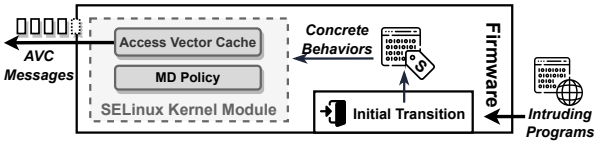


Fig. 4: Policy-driven behavior monitoring procedure in MIDAS. After the initial transition, the intruding program will be labeled, and its concrete behaviors can be monitored as AVC messages driven by the MD policy.

For design the MD policy, 1) first, we identify the processes that need to be monitored, i.e. the monitored requester of the resource. This will be used as the *scontext* item in the policy. We introduce the unified security context *suspicious.execfile* for break-in programs and *suspicious.subj* for their processes. With the type transition statements [21] dedicated to common attack entry points, MIDAS enables programs entering by these portals to be automatically labeled with *suspicious.execfile*. All the related files generated by these programs will be labeled with the same context as well. When these programs are executed, the security context of the corresponding processes can be transitioned to *suspicious.subj*. This will be used as *scontext* in MD policy rules, i.e., programs to be monitored. After such initial transition, break-in programs can be labeled with the suspicious security context, and then the program as well as its generated files are included in the monitor scope. 2) Next, to monitor suspicious behaviors, their requested resources' security contexts (i.e., *tcontext*) and their operation class (i.e., *behavior class*) can be obtained from the abstract behavior pairs in the paradigm. Overall, the key items in SELinux Policy are obtained, and thus we can implement a series of rules for malware defense, i.e. MD policy.

After MD policy is loaded in the kernel, any suspicious operation specified in this policy will trigger SELinux to generate a corresponding AVC message. With generated messages, MIDAS enables behavior monitoring of break-in programs in real-time. This procedure is independent of concrete payloads conducted by the programs, even zero-day exploitations, allowing a more general defense against malware variants.

**Homology Tracing.** Homologous processes are those derived from the same program entity. Many camouflage strategies, like self-copy or self-rename, are employed by IoT malwares to conceal themselves and decentralize malicious behaviors to submodules. For instance, the malware *Omni* can duplicate and rename itself as *ls* to reside in a covert path. The processes associated with duplicated malicious *ls* are homologous with processes of the original *Omni* binary. To fully reveal break-in program's activities under disguise, homology tracing is designed to estimate relations of processes and assemble malicious behaviors scattered among homologous processes. Namely, by aggregating homologous processes' behaviors represented as AVC messages, we can form a behavior message sequence, describing complete suspicious behaviors of the intruding program even if adopting disguises.

As shown in Algorithm 1, the homologous relations of processes can be decided based on three progressive indicator: the process identifier (i.e. *PID*), the pathname (i.e. *Path*) and the memory hash value (i.e. *Hash*). ① The *PID* refers to the ID of process that conducts the operation triggering the AVC message. Based on the parent-child relations, it can be cheaply utilized to determine the program's processes spawned through *fork(), execve() or system()* even with fake process name. However, it neglects processes created from re-execution of the same malware binary. ② Hence, the *Path* indicator is introduced to compare the exact path of the process's binary and thus fix this problem. However, although the re-executed process can be recognized by the same path of the binary even after its self-deletion, it is possible for a malware to copy or rename its binary, changing its path to achieve disguises. ③ Therefore, we further calculate the hash value of the memory-mapped file, i.e. *Hash*, to deal with other elaborate camouflage techniques. This can identify malicious processes from the identical break-in program and avoid heavy I/O costs.

The algorithm of homology tracing procedure is presented in Algorithm 1. It aggregates the input AVC messages $M$ to build an array $Q$ as output, which contains a combination of homologous process tree $t$ and its behavior message sequence with all related AVC messages *msg*. Homologous process tree is a path-splitting process tree structure to maintain each collection of homologous processes. And the behavior

message sequence contains all AVC messages triggered by the homologous processes in the corresponding tree.

**Algorithm 1:** Aggregating Behavior Trace

**Input:** Real-time AVC messages $M$
**Output:** An array $Q$ of pairs consisting of tree $t$ and message $msg$
**Data:** Created process-tree set $tSet$, collected pathname $pMap$ and hash value $hMap$ of intruding programs

```
1  Q ← ∅, tSet ← ∅
2  for msg ∈ M do
3      pid ← extract(msg)                                   // ①
4      t, pid_root ← tSet.search_PTree(pid)
5      if is_exist(t) then
6          t.add(pid)
7      else
8          path_root ← readlink_exe(pid_root)               // ②
9          hash_root ← calculate_crc32(pid_root)            // ③
10         if path_root ∈ pMap then
11             t ← tSet.get(pMap[path_root])
12             t.add(pid)
13         else if hash_root ∈ hMap then
14             t ← tSet.get(hMap[hash_root])
15             t.add(pid)
16         else
17             t ← tSet.create_PTree(pid_root) t.add(pid)
18             hMapp[path_root] ← pid_root
19             pMapp[hash_root] ← pid_root
20     Q.update(t, msg)
```

Specifically, the algorithm preferentially extracts the PID (i.e. $pid$) from each AVC message (i.e. $msg$). As the input of *searchPTree()*, the PID helps to find the homologous process tree $t$ and its root process node $pid_{root}$. Meanwhile, *searchPTree()* can recursively search and merge existing trees based on the parent-child process relations defined in */proc/$pid/stat* using path-splitting algorithm. The obtained correspondences between *PID*s and process trees are cached in *tSet* to accelerate tracing afterwards (Lines 2-4). If the required tree exists, MIDAS will add this *PID* node to the current tree $t$ (Lines 5-6). As the example shown in Figure 5.(a), the No.2720 process is a child process spawned by the parent (No.2699) of Ganiw malware to assign malicious tasks. Based on this parent-child process relation, No.2720 is merged into the green part of its homologous process tree $\beta$. Other nine processes (i.e. No.2713, No.2720, No.2727-No.2734) are forked by No.2699 as well, and are also added to the tree. Particularly, with path-splitting utilized in *searchPTree()*, Figure 5.(a) can be compressed into Figure 5.(b). Hence, homology tracing can be accelerated when searching process trees afterwards.

If the condition in Line 5 fails, MIDAS relies on the *Path* or *Hash* of the process to defeat possible camouflage(Line 8,9). Once the running process has the same pathname as any previous one, MIDAS will add this process into the process tree and confirm the homologous relation (Line 8, 10-12). The *Path* value here can be obtained by reading the symbolic link of */proc/$pid/exe*, which points to the executed binary even self-deletion happens. If the homologous relation still cannot be estimated, a lightweight half-byte CRC32 algorithm is designed to calculate *Hash* based on the first memory-mapped file in */proc/$pid/map_files*, avoiding heavy I/O. Programs with an identical *Hash* will be taken as homologous ones and added to the same process tree (Lines 9, 13-15). As shown in Figure 5, since Hash of *getty* and *l1bhr* related to No.2718 and No.2725 processes are identical with Hash of the original one *ganiw* (i.e. No.2699), the homologous relations of these processes are determined. Further, along with sub-processes spawned by these three binaries marked as respective colors in Figure 5, all the homologous processes can be combined into one process tree $\beta$. If all the conditions above are not satisfied, the process is considered from a new break-in program. After creating a new process tree for it, MIDAS will update *pMap* and *hMap* to record its *Path* and *Hash* (Lines 16-19).

Finally, this process's AVC message $msg$ will be updated to the behavior message sequence related to the obtained process tree accordingly (Line 20). As illustrated in Figure 5, the AVC messages concerned with No.2752 can be included into the sequence related to the homologous process tree $\beta$. The sequence contains AVC messages of homologous processes, including all suspicious behaviors along Ganiw's lifecycle.

On the whole, each homologous process tree can aggregate AVC messages triggered by homologues to constitute a complete behavior message sequence. In the following, it can provide a complete view of programs' behaviors to defend against possible disguised malwares.

**Behavior Identification.** For each group of homologous processes, their behaviors have been aggregated in the sequence with AVC messages. MIDAS first extracts and then summarizes the *target security context* (i.e., *tcontext*) and the *behavior class* items as abstract pairs from each message in the sequence. *tcontext* records the security context of the requested resource, while *behavior class* shows the operation category of the access. Iterating the message sequence, such procedure can obtain the behaviors conducted by the break-in homologous processes throughout their lifecycle and accomplish the conversion from concrete behaviors to abstract behavior pairs.

We give an example of the AVC message in Listing 1 in Section II, where the *tcontext* and the *behavior class* can constitute an abstract behavior pair *<file.initscriptfile, add_name>*. This pair represents a behavior that the break-in program adds files to a directory for auto-start to achieve persistence. Even if the malware forges its process name (i.e., *comm* field) to a benign program like *cat*, the *behavior class*

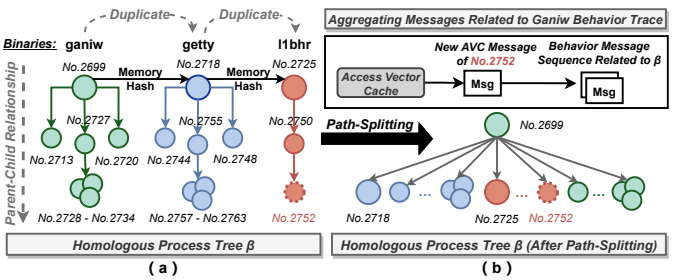


Fig. 5: Homology tracing of MIDAS when defending "Ganiw" (MD5: 0ddda3bb8590616f803a7320d890645e). Circles represent processes generated by "Ganiw". Solid lines show homologous relationships between processes while dashed lines indicate reasons for the generation of processes.

and the *target security context* can still be faithfully recorded. In addition, according to the source security context, we can also figure out that this program is not the *cat* program (with *file.execfile*), but a suspicious program from the outside (with *suspicious.subj*). More importantly, the target security context *file.initscriptfile* is a common label of directories for persistence intention. Thus even if a new variant changes the concrete path (i.e. "*/etc/rc.d*") it accesses for auto-start, the corresponding behavior message, like Listing 1, can still be triggered and extracted to the consistent abstract behavior pair.

After the above summarization, MIDAS attempts to remap the abstract behaviors to the malware lifecycle stages and estimate the maliciousness of break-in programs. Through behavioral characteristic extraction, we have established the mappings between the abstract behavior pairs <*security context, behavior class*> and the stages of the malware lifecycle in the behavior paradigm. Based on this, MIDAS can analyze each abstract pair obtained from the behavior message sequence and determine which stage of the malware lifecycle it relates to. IoT malware relies on complete lifecycle to enable a valid attack flow. Thus if behaviors conducted by a group of homologous processes are possible to constitute a malware lifecycle, the corresponding program will be identified as malicious and its impact attempts to devices will be denied. MIDAS will kill all the related processes, remove the files and all copies based on its homologous process tree and pathnames to constrain them from violating the system.

## IV. IMPLEMENTATION

We implemented MIDAS on OpenWrt with ARM, MIPS and MIPSEL architectures, which are the main targets for IoT malwares [22]. OpenWrt is one of the most popular firmwares for IoT devices and supports up to 1,981 different models [23]. It is widely used in previous works like [11], [24] and [25]. Besides, MIDAS is not limited to the three architectures or OpenWrt. It can be adapted to any Linux-based IoT device with any architecture by only recompiling the source code of MIDAS and configuring SELinux options of Linux kernel.

Specifically, the MD policy is implemented with Common Intermediate Language (CIL) based on DSSP [17] with more than 4,276 additional lines of code. In it, we design security contexts to label break-in programs, modify some system resources' security contexts, and devise rules based on the paradigm for fine-grained behavior monitoring. After compiled by *secilc*, a compiler for SELinux policy written in CIL, the binary file of MD policy can be generated. We further embedded it into the firmware and load it when booting. The homology tracing and behavior identification are implemented with C++, which analyze break-in programs' behaviors through AVC messages from */dev/kmsg* triggered by MD policy. This part runs as a background service and is protected by the MD policy from being tampered. The behavior paradigm is embedded in MIDAS' modules throughout the auditing process.

## V. EVALUATION

To present the effectiveness of MIDAS on IoT devices, we conduct thorough experiments on both benchmark malware datasets as well as real-world attack scenarios, and perform comparisons with the state-of-the-art studies. Accordingly, we answer the following research questions:

- **RQ1:** How is the performance of MIDAS to safeguard IoT devices against real-world attacks in real-time?
- **RQ2:** How is the effectiveness of MIDAS in protecting IoT devices with various hardware resources against benchmark IoT malwares?
- **RQ3:** How is the overhead of MIDAS in resource constrained IoT devices of different architectures?
- **RQ4:** How is the performance of MIDAS compared with the start-of-the-art studies?

### A. Performance in Real-World Scenarios

**Geo-location Arrangement.** To evaluate MIDAS' effectiveness against currently prevalent IoT malwares in real world, we deployed 60 virtual IoT devices with OpenWrt firmware on the public Internet for 25 days. These devices exposed customized vulnerable services, i.e. *telnet* and *ssh*, accepting any credentials submitted by attackers, thus attracting more IoT malwares. To reduce possible biases, 60 devices were equally distributed across ten countries in five continents and configured with heterogeneous architectures, as shown in Figure 6. In each country, there were two ARM, two MIPS and two MIPSEL devices. Among the two devices of each architecture in the same country, one was equipped with built-in MIDAS, and the other was not. So overall, half of devices were safeguarded by MIDAS and the other half were not.


Fig. 6: The geolocation of 60 publicly available virtual IoT devices of three architectures with or without MIDAS.

**Data Collection.** For each device, we collected login sessions, intruded malwares, logged records with their run-time behavioral information and took necessary snapshots. The size of the collected data is up to 2.88 TiB. Since the deployed devices do not provide common public services, the incoming traffic is mostly from attackers rather than legitimate users. Such idea is consistent with honeypots [11] or passive traffic monitoring systems [26]. Hence, the login session information can reflect the extent of attacks. We counted the times of each device was logined as the number of attacks and recorded their source IPs. We also backed up malwares and preserved the device status by taking snapshots. Each collected sample have been submitted to VirusTotal to determine its maliciousness, and record its family and first-found time if confirmed. Overall, the collected information help us have a closer look at the intrusion process of malwares, even previously unknown ones, and MIDAS' defense procedure against real-world attacks.

TABLE I: Results of MIDAS defending against real-world attacks. The "LOCATION" column indicates the country where the devices were located. The following columns indicates the respective architectures of the corresponding devices. Shaded rows are the results of devices protected with MIDAS, while the others show the results of devices without MIDAS. For each device, "Under attack" shows the number of attack times (Att.), the amount of source IP (IPs) where the attacks originate and the number of compromised incidents of devices (Comp.). "Malware Intrusion" shows the number of the intruding malwares (Total), unique malware amounts (Uniq.) and the number of newly-discovered malware samples (New). The last column (Comp. Ratio) shows the ratio of the total compromised incident amounts of devices with MIDAS to devices without MIDAS in each region.

| LOCATION | ARM | | | | | | MIPS | | | | | | MIPSEL | | | | | | Comp. Ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Under Attack | | | Malware Intrusion | | | Under Attack | | | Malware Intrusion | | | Under Attack | | | Malware Intrusion | | | |
| | Att. | IPs | Comp. | Total | Uniq. | New | Att. | IPs | Comp. | Total | Uniq. | New | Att. | IPs | Comp. | Total | Uniq. | New | |
| ***North America*** | | | | | | | | | | | | | | | | | | | |
| America | 2,550 | 617 | 40 | 210 | 58 | 33 | 21,091 | 1,483 | 610 | 2,261 | 194 | 123 | 6,614 | 1,415 | 142 | 720 | 155 | 92 | |
| America$^{\text{MIDAS}}$ | 6,060 | 1,372 | 0 | 255 | 89 | 18 | 48,945 | 1,308 | 3 | 423 | 118 | 75 | 23,194 | 1,362 | 0 | 518 | 146 | 65 | 3: 792 |
| Canada | 8,222 | 1,938 | 365 | 1,856 | 134 | 22 | 27,671 | 1,791 | 193 | 3,496 | 381 | 341 | 11,792 | 1,716 | 315 | 1,504 | 188 | 123 | |
| Canada$^{\text{MIDAS}}$ | 19,584 | 1,715 | 1 | 791 | 178 | 37 | 49,457 | 1,513 | 4 | 776 | 141 | 69 | 20,238 | 1,605 | 1 | 514 | 152 | 85 | 6: 873 |
| ***South America*** | | | | | | | | | | | | | | | | | | | |
| Brazil | 14,429 | 1,808 | 449 | 2,664 | 243 | 133 | 21,416 | 1,541 | 793 | 2,834 | 247 | 179 | 10,672 | 1,517 | 253 | 1,399 | 159 | 124 | |
| Brazil$^{\text{MIDAS}}$ | 17,293 | 2,171 | 2 | 748 | 243 | 115 | 20,814 | 1,713 | 0 | 477 | 117 | 43 | 8,833 | 1,573 | 0 | 297 | 99 | 55 | 2: 1,495 |
| ***Europe*** | | | | | | | | | | | | | | | | | | | |
| Britain | 19,545 | 1,651 | 985 | 5,291 | 287 | 101 | 8,624 | 1,729 | 276 | 1,420 | 125 | 64 | 10,295 | 1,590 | 391 | 1,380 | 165 | 112 | |
| Britain$^{\text{MIDAS}}$ | 15,161 | 2,135 | 1 | 666 | 230 | 68 | 9,902 | 1,735 | 0 | 777 | 209 | 89 | 29,660 | 1,296 | 0 | 387 | 113 | 63 | 1: 1,652 |
| Germany | 8,324 | 1,743 | 356 | 1,860 | 207 | 73 | 15,823 | 1,482 | 367 | 1,469 | 124 | 87 | 16,289 | 1,551 | 470 | 1,634 | 177 | 153 | |
| Germany$^{\text{MIDAS}}$ | 25,865 | 1,645 | 1 | 465 | 183 | 72 | 16,385 | 1,683 | 0 | 274 | 95 | 54 | 19,605 | 1,497 | 1 | 489 | 146 | 76 | 2: 1,193 |
| Netherlands | 2,805 | 856 | 47 | 287 | 125 | 36 | 12,500 | 1,709 | 490 | 1,818 | 177 | 122 | 7,742 | 1,459 | 134 | 768 | 124 | 67 | |
| Netherlands$^{\text{MIDAS}}$ | 10,865 | 2,387 | 0 | 730 | 237 | 107 | 14,423 | 1,767 | 0 | 561 | 153 | 92 | 10,759 | 1,698 | 1 | 396 | 158 | 104 | 1: 671 |
| ***Asia*** | | | | | | | | | | | | | | | | | | | |
| India | 17,935 | 1,883 | 1,318 | 7,215 | 381 | 116 | 7,144 | 1,464 | 265 | 994 | 131 | 68 | 4,819 | 1,093 | 31 | 449 | 110 | 63 | |
| India$^{\text{MIDAS}}$ | 24,626 | 2,051 | 1 | 844 | 254 | 109 | 12,719 | 1,616 | 1 | 324 | 92 | 31 | 17,640 | 1,608 | 1 | 599 | 136 | 74 | 3: 1,614 |
| Singapore | 7,553 | 1,574 | 261 | 1,401 | 140 | 52 | 9,579 | 1,625 | 417 | 1,659 | 125 | 78 | 12,753 | 1,602 | 341 | 1,320 | 179 | 112 | |
| Singapore$^{\text{MIDAS}}$ | 30,105 | 1,620 | 0 | 356 | 117 | 53 | 34,879 | 1,369 | 5 | 452 | 82 | 43 | 24,651 | 1,615 | 0 | 508 | 134 | 71 | 5: 1,019 |
| Indonesia | 9,766 | 1,949 | 593 | 3,162 | 229 | 108 | 10,458 | 1,781 | 412 | 2,206 | 169 | 100 | 7,211 | 1,546 | 127 | 987 | 201 | 84 | |
| Indonesia$^{\text{MIDAS}}$ | 3,852 | 1,073 | 1 | 604 | 150 | 45 | 12,846 | 1,706 | 7 | 519 | 152 | 47 | 31,585 | 1,347 | 0 | 224 | 66 | 26 | 8: 1,132 |
| ***Oceania*** | | | | | | | | | | | | | | | | | | | |
| Australia | 8,133 | 1,678 | 307 | 1,754 | 212 | 46 | 11,993 | 1,179 | 250 | 1,131 | 155 | 95 | 10,297 | 1,709 | 325 | 1,407 | 132 | 90 | |
| Australia$^{\text{MIDAS}}$ | 23,692 | 1,500 | 0 | 468 | 138 | 67 | 29,436 | 1,019 | 0 | 366 | 116 | 29 | 14,832 | 1,705 | 2 | 615 | 144 | 64 | 2: 882 |
| **Summation** | 276,365 | 22,442 | - | 31,627 | 1,382 | 572 | 396,105 | 21,109 | - | 24,237 | 1,332 | 933 | 299,481 | 20,465 | - | 16,115 | 1,058 | 704 | 33: 11,323 |

In addition, we used heartbeat mechanisms and the network monitor Snort to track traffic amount and device status. If the device loses accessibility or is used as a bot, it is considered to be compromised. The device will be reset immediately, and the source IP will be blocked to expose the device to other attackers. We use the number of times devices were compromised as metrics to evaluate the defense capability.

**Statistics Analysis.** As shown in Table I, all 60 IoT devices deployed worldwide encountered severe attacks. A total of 971,951 attack sessions reached devices for 25 days, originating from 48,805 IPs spreading over 167 countries. And we observed a total of 31627, 24237, and 16115 malwares in ARM, MIPS, and MIPSEL devices respectively. Among them, 572, 933 and 704 unique malware samples are previously unknown. It illustrates that lots of IoT malwares are active on the Internet and their attacks are surging all the time.

In the experiment, devices without MIDAS were severely disrupted. They were attacked by 2,386 different malwares in total. On average, each device has compromised for 377.4 times in 25 days. The most severely corrupted one was the ARM device in India, which was compromised up to 172 times in one day. Moreover, these devices were all used as DoS bots at least once. The most severe one, the ARM device in Brazil, has generated nearly 137Mbps attacks. Snort recorded and blocked such outbound traffic timely. Due to our traffic monitoring mechanisms, the device will be automatically reset if used as a bot, thus not damaging third-party hosts.

The status of MIDAS-protected devices, however, was quite different. As shown in Table I, despite a large amount of attacks and new variants, MIDAS can still effectively safeguard IoT devices. Specifically, MIDAS has successfully defended against 15,423 malwares. On average, each device has lost accessibility less than 1.1 times during the entire 25-day period. Compared with the most severely corrupted device in the group without MIDAS, the corresponding MIDAS-protected one, located in India with ARM architecture, only lost network accessibility once in 25 days. Meanwhile, no devices in the safeguarded group were used as bots during the whole experiment. Overall, for IoT devices with MIDAS protection, the number of compromised incidents has decreased $343.1\times$, and the length of continuous operation time is $179.2\times$ greater than the group without MIDAS on average. This demonstrates that MIDAS can successfully defend against the latest variants in real-world timely, based on the robust behavior paradigm.

For compromised cases, we analyzed the logs including AVC messages and related samples to depict the attack process, and restored the snapshots to identify the compromise reason. In the MIDAS-protected group, compromised situations

were judged to lose availability due to severe external DDoS attacks, further causing network instability and heartbeat packet loss. Thus these MIDAS-protected devices were threatened by DDoS attacks on the Internet rather than intruding IoT malwares. Conversely, for devices without MIDAS, 58.6% of the compromised incidents were caused by accessibility loss, including the critical service termination, severe corruption to the file system, etc., of which only 0.5% were affected by the external DDoS attack mentioned above (34 times). And 41.4% of compromised incidents were deemed to be used as bots.

In addition, we cross-checked malwares intruding into the devices with MIDAS and the ones without MIDAS. Among them, 443, 746 and 389 malwares only attacked devices without MIDAS but were not found in the protected ARM, MIPS and MIPSEL devices respectively. We thus executed these malwares in MIDAS-protected IoT devices. Results showed that MIDAS could defend against all of them, while the original devices without MIDAS were compromised 969 times because of these malwares. On the other hand, there were 244 malwares attacking MIDAS-protected devices but not on the other group. We further executed these malwares inside a device without MIDAS, and found that 103 of them can compromise the device, while none of them successfully compromised the MIDAS-protected hosts in the experiment. Such cross-validation fully demonstrates that devices with built-in MIDAS can effectively defend against attacks from identical IoT malwares compared to the other group.

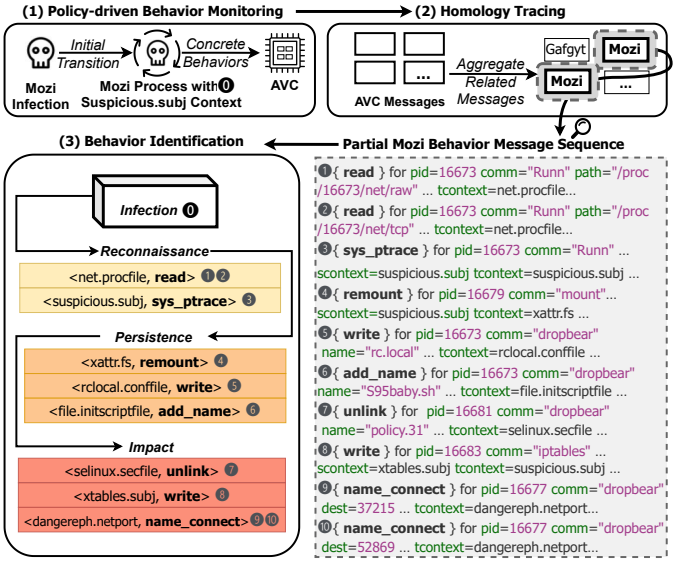


Fig. 7: MIDAS' defense procedure against the *Mozi* variant, i.e. mozi.a (MD5: eec5c6c219535fba3a0492ea8118b397).

**Case Study:** MIDAS has successfully defended 1,019 unique malwares under 627,906 times of attacks over the 25 days. Among them, the IoT malware *Mozi* enjoyed a huge surge in the world [27]. We take it as an example to illustrate the behavior auditing procedure of MIDAS. Driven by MD policy, MIDAS monitored *Mozi*'s behaviors in real-time after initial transition from *wget* entry point, and recorded as AVC messages shown in Figure 7.(1). In this example, *Mozi* first gathered network information from */proc/$pid/net/raw* and */proc/$pid/net/tcp* at the reconnaissance stage (❶-❷). It then applied an anti-debug trick, tracing itself to detect debugger since one process can only have one tracer at a time (❸). Meanwhile, *Mozi* remounted the "*/overlay*" file system with read/write permissions (❹), camouflaged its process name to "*dropbear*" and created scripts for auto startup in multiple ways for persistence (❺-❻). Finally, it attempts to achieve impacts, trying to damage the secure mode environment (❼), tamper iptable rules (❽) and execute payloads targeting vulnerabilities like CVE-2017-17215 for proliferation (❾-❿). Although these behaviors were conducted by many submodules, i.e. different processes from distinct re-executions, MIDAS can still aggregate them into one message sequence of the same homologous process tree, as shown in Figure 7.(2), After the message sequence was abstracted, MIDAS then utilized these abstract pairs to identify the malware lifecycle based on the behavior paradigm as shown in Figure 7.(3). Obviously, none of the sub-process could assemble the complete stages of malware lifecycle alone, which indicates the necessity of homology tracing. Finally, all the homologous processes were killed, and the related binaries were removed, thus constraining it from destruction and spreading.

According to VirusTotal, this variant was first discovered on June 17, 2021, and is one of the most heavily mutated active versions from the original one. In general, suffering threats from the latest variants under various disguises, MIDAS can still audit malware behaviors and achieve effective defense.

### B. *Effectiveness on Benchmark Malwares*

To further evaluate the effectiveness of MIDAS, we conducted experiments on benchmark dataset composed of 115,970 IoT malwares distributed in 28 families. The evaluation is performed on 24 virtual IoT devices with distinct hardware in terms of architectures, CPU models and memory sizes to verify the performance on devices with different resources, as shown in Figure 8.

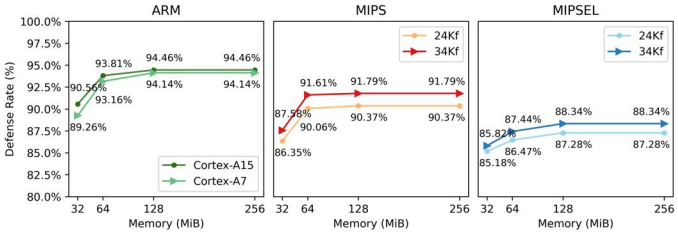


Fig. 8: The defense rates of MIDAS on IoT devices with 24 different hardware resources.

We put each malware of the datasets into a MIDAS-protected virtual device and executed it. If the device meets any of the following conditions, we assume that the device has been compromised: (1) lose accessibility, (2) be used as a DoS bot, (3) the malware process still exists in a limited time (we select five minutes as the limit referred to [15]), (4) the malware persists after a reboot. Otherwise, we consider MIDAS to have successfully defended the malware. After each test, we reset the device to ensure environment consistency. Finally, we calculated successfully defended malwares on each device.

Results are presented in Figure 8. It can be seen that on ARM, MIPS and MIPSEL devices, MIDAS can defend against up to 94.46%, 91.79% and 88.34% of the benchmark samples, respectively. This illustrates that MIDAS can effectively defend

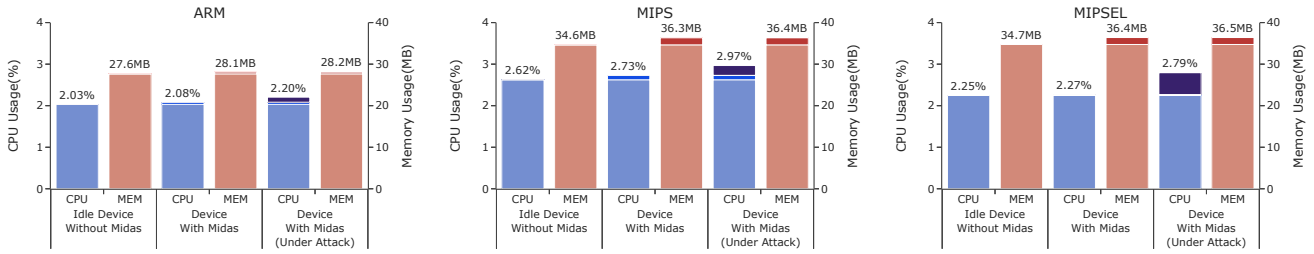


Fig. 9: The CPU and memory usage of MIDAS on IoT devices with three architectures under different circumstances. The first "Idle Device" group indicates the hardware usage when the device is fully booted and not installed with MIDAS. Increments in the second and third groups represent the overhead of MIDAS in the case of devices with or without malware attacks.

against malwares on different architectures. We have carefully analyzed unsuccessful defense cases and found that the devices were judged to be compromised because of the third criterion (i.e., malware process exists over 5 minutes). We found that some C&C servers required for these malwares were no longer accessible. Some malwares, like *Gafgyt* variants, will not attempt to cause malicious impacts to the device but constantly try to connect or sleep when they cannot connect to the C&C server. Thus MIDAS could not identify the malwares without behaviors of its entire lifecycle in 5 minutes. However, it also means that the C&C server of these variants is no longer active and cannot pose a severe threat in actual attacks. In fact, malwares with partial lifecycle are either impossible, as they lack key steps to penetrate a device, or cannot cause any serious harm, thus they do not affect IoT devices' functionality. This situation only appears when malwares have become dormant and cannot accomplish their whole intentions. Once these malicious programs become active and try to damage the device, they will be stopped by MIDAS.

Meanwhile, as can be seen from the figure, the CPU and memory resources do not have a significant influence on MIDAS' performance. The average variation of MIDAS' defense rate is only 1.53% on devices with various hardware resources. We further analyzed the reasons for such variation specially when memory resources are limited under 32 MiB. With such hardware, some malware samples' execution was severely affected and could not sufficiently expose their behaviors within the time limit. Since MIDAS estimates programs' maliciousness based on their behavioral attempts, such situations might affect the experiment results in 5 minutes.

### C. Overhead of MIDAS

For constrained IoT devices, it is crucial to ensure that security mechanisms are lightweight. To evaluate its overhead, we measured the CPU and memory usage of the system under three conditions: (1) without MIDAS protection, (2) with MIDAS protection but without malware infection, (3) with MIDAS and currently defending against malware attacks.

Experiments were conducted on an ARM Cortex-A15 virtual device, a MIPS 24Kf virtual device, and a MIPSEL 24Kf virtual device. The selected CPUs were released before 2012 and the performance are relatively limited compared to recent ones, thus demonstrate MIDAS' overhead on various IoT devices. The hardware usage is collected every five seconds for a continuous period of five minutes after the system is fully booted. To evaluate the overhead on the third condition mentioned above, we also executed each benchmark sample and measured the system hardware usage every five seconds during five minutes. This five-minute time was chosen to be consistent with RQ2 and the five-second interval avoids bringing extra resource costs for too frequently sampling.

The average CPU and memory usage is presented in Figure 9. Results show that MIDAS brings minimal overhead to all the tested devices. Take the ARM device as an example. The overall CPU usage is 2.03% after the system is fully booted, with 27.6 MiB of memory consumption. Whereas, after MIDAS is started, the CPU usage has only increased to 2.08%, with additional 0.5 MiB of memory used by MIDAS. When defending against malwares, MIDAS only brings additional 0.17% CPU and 0.6 MiB memory consumption.

Various mechanisms of MIDAS are dedicated to IoT devices for reducing overhead. Specifically, (1) monitoring behaviors based on triggered AVC messages avoids bringing significant run-time costs, (2) the homology tracing applies a progressive strategy, prioritizing more lightweight indicators against disguise, and (3) to avoid heavy I/O costs, the half-byte CRC32 algorithm is used to calculate the hash based on already loaded memory files. It takes only 64-bytes for look-up table, but performs 3× faster than the bitwise version, achieving balance between overhead and cryptographic accuracy.

### D. Comparison with Related Studies

We further compare MIDAS' defense effectiveness and false positive issues with the IoT malware *defense* research HADES-IoT [28] and the IoT malware *detection* framework proposed by Li et al. [29]. HADES-IoT monitors process spawning and stops processes outside the whitelist. To the best of our knowledge, HADES-IoT is the only state-of-the-art research that can achieve on-device IoT malware defense. The framework of Li et al. uses graph neural network (GNN) to detect IoT malwares through function call graphs. Since it cannot directly run on resource-constrained IoT devices as MIDAS and HADES-IoT do, we can only evaluate it on a PC with 64 GiB of memory and an Intel i7-10700 processor. Meanwhile, this framework only supports malware detection without malware discovery and further defense (i.e. malware constraining), thus we can only measure its detection rate instead of defense rate. Results are presented in Table II.

For fair comparison, we examined the effectiveness of the aforementioned works on the malware dataset proposed by HADES-IoT. It can be seen that MIDAS successfully defended against all the malware samples in the dataset and achieved the same defense rate as HADES-IoT presented in its paper. However, the framework of Li et al. only detected 77.78%

TABLE II: Comparison results of the defense effectiveness and false positive rates between MIDAS, HADES-IoT and GNN-based framework proposed by Li et al. [29].

| Comparison | Defense Rate *Malware Dataset* | False Positive Rate *Application Dataset* |
|---|---|---|
| MIDAS | 100% | 0% |
| HADES-IoT [28] | 100% | 44.23% |
| GNN [29] | 77.78%* | 28.85% |

Note: * represents the detection rate of the framework, since it can only achieve malware detection without further defense (i.e., malware constrain).

of malwares in the dataset, where it misses the remaining malwares that use obfuscation.

Also, we evaluated false positive rates on all the applications embedded in the OpenWrt firmware. MIDAS reported no false alarms toward these applications. On the contrary, HADES-IoT uses a rigid whitelist comparison mechanism, which only allows executions of processes in a fixed list collected in one hour after boot. We follow the same procedure to collect the allow list of HADES-IoT to estimate its false positive rate, and found that 44.23% of the applications are blocked by HADES-IoT. Such a rigid method causes high false positive issues, and thus is unscalable and inapplicable for IoT devices in the real world. The framework of Li et al. has 28.85% false positive rate towards benign applications. We found that it misclassifies some applications for essential system functionalities such as *procd*. Since these applications involves much complex call relations, they are likely to be falsely reported by the framework based on function call graph features.

## VI. DISCUSSION AND LIMITATION

**Practicality on physical IoT devices.** Low-end IoT devices using proprietary operating systems, unlike those using Linux, cannot be adapted to MIDAS. MIDAS is applicable for Linux-based IoT devices with hardware resources beyond Class-2, as defined in RFC7228 [30]. For these devices, adapting MIDAS only requires minor modifications to the firmware. We successfully tested MIDAS on devices like PSG1218, Redmi AX6S and Xiaomi 4A, whose details are provided in the supplementary materials[1]. MIDAS can bring the considerable defense capabilities to these physical devices while occupying less than 0.68% CPU and 1.86 MiB memory. Therefore MIDAS is practical for real-world IoT device deployments.


**(a)** *PSG1218 with Midas* **(b)** *Redmi AX6S with Midas*

Fig. 10: Practicality of MIDAS on Physical Devices

**Defense against adaptive attacker.** MIDAS constitutes a significant challenge for adaptive attackers to bypass because of the behavior paradigm. Instead of relying on specific patterns, MIDAS utilizes abstract representations of malware behaviors. As long as the malware's malicious intention remains stable, the security sensitivity of the accessed resources will remain relatively constant, thus the abstract behavior pairs that correspond to the paradigm are unchanged. For extreme cases where the paradigm is allegedly bypassed, the basic integrity of the firmware still cannot be corrupted, because the MD Policy is still enforced and essential resources, such as u-boot partitions, are still protected. However, since MIDAS' defense is based on the security mechanism provided by the kernel, therefore, if the kernel itself is corrupted, attackers may bypass MIDAS' defense and compromise devices.

**Generality of the defense.** In IoT devices, only fixed programs are used as entrances by malwares, such as uHTTPd or Wget, but specific vulnerability exploitations are changing. Based on this observation, MIDAS' defense are designed to be independent of specific vulnerabilities used by malwares, but includes threats from attack entrances into the auditing scope. In this way, even if malwares exploit zero-day vulnerabilities for infection, MIDAS can still defend against these threats.

**Time Delay of Behavior Auditing.** Timely auditing malwares' behaviors is important for MIDAS to achieve effective defense. Therefore, we further evaluated the delay between the behavior conduct time of the program and the behavior audit time of MIDAS. In our dataset, samples perform an average of 4 malicious operations per second. Under such circumstances, the average time delay is only 0.04s. Additionally, a *Tsunami* variant conducts the most frequent malicious behaviors in our dataset, with 398 operations per second. In such worst case scenario, MIDAS' time delay is only 0.89s. This fully demonstrates the low latency of MIDAS.

## VII. RELATED WORK

**Malwares on IoT devices.** Many researchers have conducted surveys on IoT malwares. For example, Alwari et al. [4] investigated IoT malware lifecycle and point out that insufficient security mechanisms make IoT malwares a severe threat. Antonakakis et al. [31] focused on the specific IoT malware, Mirai and presented its evolution process. Studies like these give us important lessons for extracting malware behaviors. Differently from these prior works, we not only analyze IoT malwares, but also propose a generic and lightweight safeguard framework equipped with abstract behavior paradigm to defend against IoT malware attacks.

**Malware Detection.** Many studies focus on detecting malwares on various platforms. Research like [6] [32] extracts concrete features, like CFG, opcode or gray-scale image to detect IoT malwares. Some research like [8] [33] designs anti-malware framework for PC or mobile domains. Different from MIDAS, studies like these cannot be applied to IoT devices for malware defense due to the following reasons. Firstly, most studies like [6] [32] focus on malware detection to separate malwares from benign programs based on concrete features, without on-device malware discovery and further resistance. On the contrary, MIDAS can track the complete lifecycle of IoT malwares from the intrusion to impact in the device and achieves effective defense with the behavior paradigm. It also blocks ongoing attacks and thus safeguards the firmware. Secondly, many frameworks like [8] [33] are inapplicable due to IoT devices' limited hardware resources and heterogeneous architectures. Methods like machine learning take up too much storage and computational resources to be embedded in IoT

devices. Approaches such as traffic monitoring [34] needs to be placed externally, suffering from inadequate dimensions for malware analysis and device protection, like files or processes. However, MIDAS can achieve lightweight on-device malware defense on multiple aspects for heterogeneous IoT devices.

**Malware Defense.** Researchers have proposed various methods to defend against malwares. For example, FlashGuard [35] is a firmware-level recovery system that can defend against ransomware attacks on SSD. Zhang et al. [36] proposed Scarecrow, a deception engine for environment camouflaging to deactivate evasive malwares. However, these methods are designed to defend against PC malwares, with relatively high hardware requirements and lack of compatibility, which cannot be easily applied to heterogeneous resource-constrained IoT devices. Recently, there are few studies focusing on IoT malware defense. For example, HADES-IoT [28] collects a fixed list of benign processes as whitelist to conduct process comparison and denies execution of any other program. Compared with MIDAS, such mechanism brings high false alarm rate and hinders the scalability of IoT devices.

## VIII. CONCLUSION

This paper presents MIDAS, an adaptive safeguard framework for Linux-based IoT devices with heterogeneous architectures by real-time behavior auditing. Through sensitive behavior acquisition of 115,970 IoT malwares, we build a robust and abstract hierarchical behavior paradigm to defend mutating malwares in further behavior auditing. During auditing, MIDAS monitors behaviors of break-in programs with the built-in MD policy, traces operations of homologous submodules under disguises, and summarizes abstract behaviors to unveil malware lifecycle and further constrain them. Our evaluation shows that MIDAS can effectively safeguard IoT devices when suffering from 971,951 real-world attacks. The number of compromised incidents of devices with MIDAS decreases $343.1\times$, and the continuous operating time is $179.2\times$ longer than devices without MIDAS. Besides, MIDAS successfully defends against up to 94.46% of the benchmark malwares with less than 1.8MiB of memory and 0.54% CPU usage.

## REFERENCES


[1] E. IoT, "Iot developer survey key findings," https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2020.pdf, 2020.

[2] I. Security, "Ibm x-force threat management for iot," https://www.ibm.com/downloads/cas/GBPQEPY1, 2020.

[3] Qihoo, "Iot_reaper: A rappid spreading new iot botnet," https://blog.netlab.360.com/iot_reaper-a-rappid-spreading-new-iot-botnet-en/.

[4] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monrose, and M. Antonakakis, "The circle of life: A Large-Scale study of the IoT malware lifecycle," in *USENIX Security 21*, 2021, pp. 3505–3522.

[5] "A survey of iot malware and detection methods based on static features," *ICT Express*, vol. 6, no. 4, pp. 280–286, 2020.

[6] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of iot malware based on image recognition," in *COMPSAC 2018*, vol. 02, 2018, pp. 664–669.

[7] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: Evaluating android anti-malware against transformation attacks," ser. ASIA CCS '13. New York, NY, USA: ACM, 2013, p. 329–334.

[8] L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis," in *USENIX Security 12*. USENIX Association, Aug. 2012, pp. 569–584.

[9] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *OSDI 2010)*, Vancouver, BC, 2010.

[10] VirusTotal, 2021. [Online]. Available: https://www.virustotal.com/

[11] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of iot compromises," in *WOOT 15*, Washington, D.C., 2015.

[12] Vx-underground, "vx-underground," 2021. [Online]. Available: https://www.vx-underground.org/

[13] ArditDulemata, "ArditDulemata/Curated-Malware-Database," 2020. [Online]. Available: https://github.com/ArditDulemata/Curated-Malware-Database

[14] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *RAID 2016, Paris, France, September 19-21, 2016, Proceedings*, ser. Lecture Notes in Computer Science, F. Monrose, M. Dacier, G. Blanc, and J. García-Alfaro, Eds., vol. 9854. Springer, 2016, pp. 230–253.

[15] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 161–175.

[16] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, D. Zamboni, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 108–125.

[17] Defensec, "dssp SELinux Policy," 2021. [Online]. Available: https://git.defensec.nl/

[18] J. Security, "Automated malware analysis - joe sandbox cloud basic," https://www.joesandbox.com/#linux, 2021.

[19] T. M. Corporation, "Tactics - enterprise — mitre att&ck®," https://attack.mitre.org/tactics/enterprise/, 2013.

[20] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy," *Technical report*, 2018.

[21] S. Project, "Nb domain and object transitions - selinux wiki," https://selinuxproject.org/page/NB_Domain_and_Object_Transitions, 2015.

[22] P. Corporation, H. Y. Lin, and Y. Osawa, "Understanding the iot threat landscape and a home appliance manufacturer's approach to counter threats to iot," 2019.

[23] The OpenWrt Project, "OpenWrt - Supported devices," 2021. [Online]. Available: https://openwrt.org/supported_devices

[24] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. A. Chen, T. Xu, Y. Chen, and J. Yang, "Understanding fileless attacks on linux-based iot devices with honeycloud." New York, NY, USA: ACM, 2019.

[25] S. Sundaresan, S. Burnett, N. Feamster, and W. de Donato, "BISmark: A testbed for deploying measurements and applications in broadband access networks," in *USENIX ATC*, 2014, pp. 383–394.

[26] D. Moore, C. Shannon, G. M. Voelker, S. Savage *et al.*, *Network telescopes: Technical report*. Department of Computer Science and Engineering, University of California ..., 2004.

[27] T. Seals, "Mozi botnet accounts for majority of iot traffic — threatpost," https://threatpost.com/mozi-botnet-majority-iot-traffic/159337/, 2020.

[28] D. Breitenbacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer, and Y. Elovici, "Hades-iot: A practical host-based anomaly detection system for iot devices." New York, NY, USA: ACM, 2019.

[29] C. Li, G. Shen, and W. Sun, "Cross-architecture internet-of-things malware detection based on graph neural network," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–7.

[30] C. Bormann, M. Ersue, and A. Keränen, "Terminology for Constrained-Node Networks," RFC 7228, May 2014.

[31] M. Antonakakis, T. April, M. Bailey, and M. B. et al., "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, 2017, pp. 1093–1110.

[32] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen, "Efficient signature generation for classifying cross-architecture iot malware," in *CNS 2018*, 2018, pp. 1–9.

[33] Z. Zhu and T. Dumitraş, "Featuresmith: Automatically engineering features for malware detection by mining the security literature." New York, NY, USA: ACM, 2016.

[34] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, 2016, pp. 807–822.

[35] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, "Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware," ser. CCS '17. ACM, 2017, p. 2231–2244.

[36] J. Zhang, Z. Gu, J. Jang, D. Kirat, M. Stoecklin, X. Shu, and H. Huang, "Scarecrow: Deactivating evasive malware via its own evasive logic," in *DSN 2020*, 2020, pp. 76–87.