An Enclave Assisted Snapshot-based Kernel Integrity Monitor

Dimitris Deyannis*
FORTH-ICS
Heraklion, Greece
deyannis@ics.forth.gr

Dimitris Karnikis*
FORTH-ICS
Heraklion, Greece
dkarnikis@ics.forth.gr

Giorgos Vasiliadis
FORTH-ICS
Heraklion, Greece
gvasil@ics.forth.gr

Sotiris Ioannidis
FORTH-ICS
Heraklion, Greece
sotiris@ics.forth.gr

ABSTRACT

The integrity of operating system (OS) kernels is of paramount importance in order to ensure the secure operation of user-level processes and services as well as the benign behavior of the entire system. Attackers aim to exploit a system's kernel since compromising it provides more flexibility for malicious operations compared to compromising a user-level process. Acquiring access to the OS kernel enables malicious parties to manipulate process execution, control the file system and the peripheral devices and obtain security- and privacy-critical data. One of the most effective countermeasures against rootkits are kernel integrity monitors, implemented in software (often assisted by a hypervisor) or external hardware, aiming to detect threats by scanning the kernel's state. However, modern rootkits are able to hide their presence and prevent detection from such mechanisms either by identifying and disabling the monitors or by performing transient attacks.

In this paper we present SGX-Mon, an external kernel integrity monitor that verifies the operating system's kernel integrity using a very small TCB while it does not require any OS modifications or external hardware. SGX-Mon is a snapshot-based monitor, residing in the user space, and utilizes the trusted execution environment offered by Intel SGX enclaves in order to avoid detection from rootkits and prevent attackers from tampering its execution and operation-critical data. Our system is able to perform scanning, analysis and verification of arbitrary kernel memory pages and memory regions and ensure their integrity. The monitored locations can be specified by the user and can contain critical kernel code and data. SGX-Mon scans the system periodically and compares the contents of critical memory regions against their known benign values. Our experimental results show that SGX-Mon is able to achieve 100% accuracy while scanning up to 6,000 distinct kernel memory locations.

CCS CONCEPTS

• **Security and privacy → Operating systems security**.

KEYWORDS

Intel SGX, Linux kernel, integrity monitor, secure enclaves

*Also with the Department of Computer Science, University of Crete, Greece

ACM Reference Format:
Dimitris Deyannis, Dimitris Karnikis, Giorgos Vasiliadis, and Sotiris Ioannidis. 2020. An Enclave Assisted Snapshot-based Kernel Integrity Monitor. In *3rd International Workshop on Edge Systems, Analytics and Networking (EdgeSys '20), April 27, 2020, Heraklion, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3378679.3394539

1 INTRODUCTION

One of the most crucial security problems is to ensure the integrity of the Operating System (OS), as it has a direct effect to the security of its running processes and their data. Modern kernels found in end-user devices and servers consist of millions lines of code and typically contain vulnerabilities that are too difficult to identify or debug. At the same time, adversaries and attackers are becoming more and more powerful, by utilizing years of experience and advanced tools, creating attacks that stay under the radar of modern anti-malware systems. Particular families of threats, such as transient rootkits, repetitively tamper critical memory regions of the kernel for a very short period of time and then restore them to their original state, rendering themselves virtually undetectable. Moreover, advanced rootkits try to identify and disable security monitors prior to performing their malicious activities in order to avoid hide their presence in the system.

One of the main defences against such kind of state-of-the art kernel-side attacks include kernel data and code integrity monitoring techniques. Integrity monitors aim to ensure that the state of the OS and its environment remain intact after a secure boostrap-phase from malicious software and users. Usually, these monitors are triggered in very short time intervals or by certain event patterns [7], scan selected memory regions, such as security-crucial kernel memory pages, and try to validate that the contents of these pages have not been tampered. In general, there are two major approaches to modern kernel integrity monitors, *hardware-assisted* [9–11, 13] and *hypervisor-assisted* [4, 14, 16, 17]. The hardware-assisted monitors can be supported by either an additional chip connected to the memory bus of the system, responsible to snoop the selected memory regions, or by an external device, responsible for analyzing the state of the operating system and performing the necessary checks. Apart from successfully identifying kernel-side attacks, modern integrity monitors have to also ensure that they remain undetected and unaffected by advanced rootkits and are able to respond when malicious activities are discovered.

Intel SGX [1] is a security mechanism provided by Intel in modern x86 CPUS that extends the ISA with instructions that enable trusted execution. SGX acts as a reverse sandbox, meaning that no other process, user or even the OS kernel can access its memory regions, except for the trusted application. Utilizing the properties and guarantees provided by SGX in combination with integrity monitoring can lead to the development of advanced monitors that are able to hide their presence in the system and protect the integrity of their code and data, remaining unaffected from rootkits that aim to disable them.

In this paper, we present a novel snapshot-based user space kernel integrity monitor, namely SGX-Mon, that encloses its monitoring mechanism and state inside SGX enclaves, able to identify transient rootkits while remaining untampered and undetected by attackers. This paper's contributions are the following:

- The first, to our knowledge, kernel integrity monitor that leverages Intel SGX in order to protect its code and data from identification and modification. SGX-Mon utilizes a very small TCB, able to be contained in a single SGX enclave and is easily audited.
- SGX-Mon is purely software-based, running in the user space, and does not require external or non-commodity coprocessors, devices, TPMs or hypervisors. This restricts potential memory leaks and vulnerabilities that are introduced by extra hardware or complex software, such as hypervisors, that might put the kernel and other sensitive memory regions at risk.
- An evaluation of the effectiveness of our system in identifying transient kernel-side attacks as well as a study of the appropriate monitoring intervals that guarantee that transient rootkits are not able to perform any malicious actions and still be able to remain undetected.

2 BACKGROUND

Intel Software Guard Extensions (SGX) [1, 5] is a group of security instructions offered by modern Intel x86 CPUs, firstly introduced with the Skylake family of processors. These instructions provide secure and hardware-assisted isolated software containers, called *enclaves*, whose code and data cannot be read or modified by any other process aside from the one utilizing them. This reverse-sandbox property also restricts the OS kernel or debuggers from reading or tampering their contents. Intel SGX utilizes hardware-assisted encryption, performed by the CPU, using a supplied on-chip mechanism called Memory Management Engine (MME). The MME is responsible for encrypting a portion of the live memory and providing it to SGX where the data and the code of several enclaves may reside. Data from the trusted enclave are decrypted on the fly during execution within the CPU and are accessible only by the enclave. When enclave memory pages need to be swapped out and moved to the DRAM, SGX encrypts them using the MME. As a result, every process is restricted from accessing the enclave contents. The available live memory of Intel SGX enclaves ranges between 64MB and 128MB and is defined by BIOS settings. However, this does not limit the developer from accessing more memory by utilizing swapping. Also, enclave data can be securely sealed and exported in the untrusted file-system in an encrypted format, accompanied by metadata used for integrity checking upon reuse.

A typical SGX application consists of two parts: (i) the untrusted application that resides in the unstrusted OS and communicates with the enclave and (ii) the secure enclave that may be bound to one or multiple applications. Communication between those two is achieved by specific functions and APIs that are declared in SGX

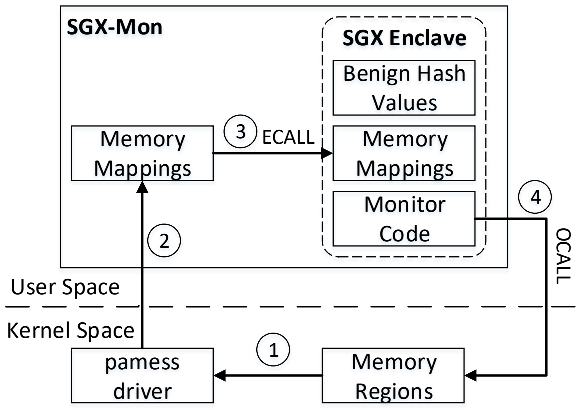

Figure 1: Architecture overview

Enclave Definition Language (EDL) during the software development and cannot be modified or extended after compilation and enclave signing. Enclaves are prohibited from directly performing undeclared I/O, accessing any system calls, or invoke privileged instructions since the host OS kernel cannot be trusted and is rendered inaccessible. The developer has to proxy such requests to the untrusted part of the application. Such calls, known as OCALLs, transfer the execution outside of the secure enclaves and can only be invoked by the enclaves. Similarly, an application can perform ECALLs, which transfer the execution from the untrusted application to the trusted enclave, in predefined entry points, invoking a predefined enclave function. Both ECALLs and OCALLs are defined in the EDL file during the application's development and can not be modified afterwards.

3 DESIGN AND IMPLEMENTATION

Our kernel integrity monitor puts its security monitor, which represents its entire Trusted Computing Base, in an SGX enclave. Hence, the security monitor is safe from attacks that can potentially compromise the Linux kernel and affect its execution. Moreover, attackers can not inspect its code and identify that SGX-Mon exists in the system, scanning the kernel for modifications. In addition, we use techniques to deprive an attacker from modifying the OS kernel in order to prevent running unauthorized code on the target system, as well as stop attacks that involve modifying the system memory layout, e.g. through changing virtual memory mappings. This is an important step toward complete security protection of the kernel.

In its core, the integrity monitor is external, snapshot-based, that can also provide programmability and easy deployment. During the secure bootstrap of the system, SGX-Mon obtains the benign values of selected kernel memory regions that are not expected to be modified during normal execution. Then, it hashes these values and securely stores them inside the SGX enclave. During system execution, SGX-Mon periodically rescans these regions, computes its hash values and compares them with the benign ones. If a value is found to be modified, the system reports the existence of

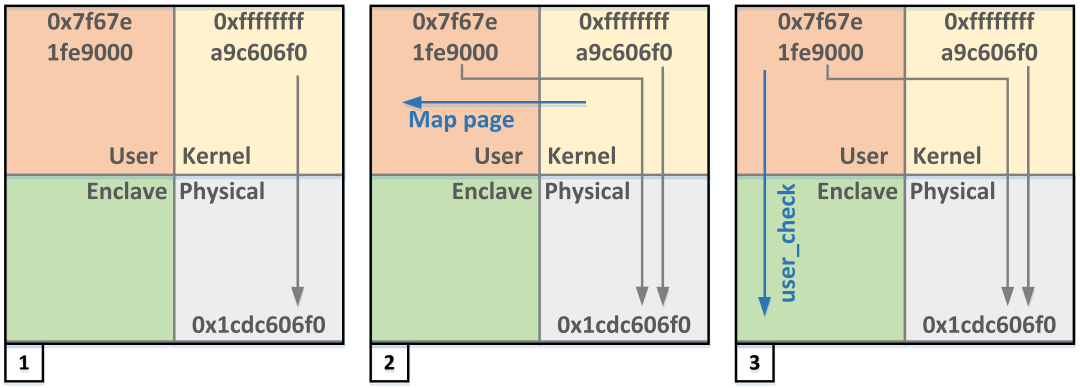

Figure 2: Mapping OS kernel memory to the address space of the integrity monitor. In step 1 we locate a desired kernel virtual address pointing to a physical address. In step 2 we duplicate this mapping to user space using our page table manipulation kernel module. In step 3 we pass the user space virtual address to the SGX-enclave using the *user_check* option.

a possibly malicious action. Besides the static text parts of the kernel or the already loaded LKMs that can easily hashed, the OS kernel consists of additional parts that frequently change; for example the VFS layer's data structures change when new file-systems are mounted or removed. Also, every LKM can add function pointers. The memory regions to be monitored can be specified by the user, and can include pages that contain kernel text, loadable kernel modules, or function pointer arrays (i.e., jump tables). A graphical representation of our system's architecture is depicted in Figure 1.

3.1 Mapping kernel memory to SGX enclaves

During bootstrapping, our integrity monitor needs to acquire the kernel memory regions that need to be monitored. Since these regions are located in the kernel virtual address space, the first step is to map them to the address space of the user process that issues the execution of the secure enclave that performs the monitoring.

In order to provide the desired integrity monitoring functionality from the user space we need to develop a mechanism to reference kernel memory regions. The Linux kernel prohibits user space applications to directly access memory regions that have not been assigned to them. Typically, all memory accesses to pages that are not mapped to the process's virtual address space result in a segmentation violation since they are considered illegal. For this reason the memory regions where the kernel and its data structures reside have to be mapped to the integrity monitor's address space. In order to bypass this OS functionality without modifying the operating system kernel we developed a loadable kernel module, named *pamess*, able to map user-specified memory regions to user space. These regions can correspond to pages that contain kernel function pointers, data, as well as LKM or kernel text. The virtual addresses of the aforementioned memory locations are obtained via the */proc/kallsyms* interface. In our case, the Linux kernel under test is built with *CONFIG_KALLSYMS=y* and *CONFIG_KALLSYMS_ALL=y* so no recompilation of the kernel is required. However, these two options are not a strict requirement for the proper operation of our system.

Since the kernel symbol lookup table is present, we are able to simplify the development in two ways. Firstly, no custom memory scanner is required in order to locate all the kernel memory regions subject to monitoring. Secondly, it allows us to easily locate the address of the kernel page table by acquiring the address of the *init_mm* symbol without explicitly exporting it via modifying the kernel. In this case, no modifications to the host's operating system are required and our system is able to operate at its full potential just by loading our custom kernel module. In scenarios where the access to the kernel lookup table has to be restricted, we would locate the various memory locations using either an external symbol table or via a custom memory pattern scanner. The *pamess* kernel module is only required during the secure-bootstrap phase, in order to acquire the correct memory mappings, and can be unloaded afterwards as it is not needed during SGX-Mon's execution.

The process of making the aforementioned memory pages available to the SGX enclave is depicted in Figure 2. During a secure bootstrap phase we allocate a data structure inside the SGX enclave responsible for storing all the available data and metadata of each kernel region, such as the correct checksum, number of required pages, offsets, etc. Our kernel loadable module resolves the physical mapping of each desired kernel virtual address and instructs the integrity monitor to allocate the appropriate amount of pages in its address space, as shown in step 1. Then, in step 2, the module makes the allocated page to point to the same page as the kernel virtual address by duplicating the PTE in the user-page table. Finally, in step 3, we update the data structure inside the enclave by passing a pointer to the virtual page of the monitor's address space using the *user_check* option. By issuing read operations of the appropriate size the enclave side of the monitor is now able to inspect the contents of the desired kernel memory regions.

3.2 Kernel integrity monitoring inside the SGX

The process of monitoring the specified memory locations is entirely performed inside the SGX enclave. The first part of process is acquiring the correct checksums of the specified memory regions. During the secure bootstrap phase, once the page mapping process is finished, the monitor obtains the hashes of the contents of all specified locations and stores them in its data structures. Since this operation is performed during a secure bootstrap phase, the state of the entire system is considered safe, thus these checksums are considered a reference point for the security of the system.

After the process of obtaining the checksums is finished we assume that the secure bootstrap phase has finished and the system could be put at risk at any time. The monitor operates as a daemon, running in an infinite loop, constantly iterating through all the memory regions hashing their contents. The hashes are compared for changes against the values obtained during the previous phase. Since the entire checksumming process, as well as the clean-state checksums, reside inside the SGX enclave, the monitoring operation remains untampered.

The system is able to perform the checksumming operations by using either the CRC-32 or the SHA-256 hashing algorithm. In its default operation, the monitor opts for CRC-32, implemented in accordance with ISO 3309. We choose CRC-32 mainly due to its speed, simplicity and capability to cover a large number of individual memory locations without severe performance overhead that penalises the safety of the system (see Section 4 for performance comparison). The system is also able to perform the monitoring by entirely using SHA-256 in scenarios where CRC-32 is not considered sufficient enough to provide collision free results. However, SHA-256 is significantly slower and while it offers higher security it is not able to cover the same amount of kernel memory regions in the same frequency. In such cases, the integrity monitor can be tuned to use CRC-32 for the majority of memory locations and SHA-256 for regions that are regarded to be highly security critical.

4 EVALUATION

In this section we present the evaluation of our SGX-enabled Kernel Integrity Monitor in terms of performance and accuracy. We explore the characteristics of CRC-32 and SHA-256 algorithms in respect to the detection rate of a kernel side attack. We choose to evaluate SGX-Mon with CRC-32 and SHA-256 as the former stands as a good representation of the fast hash collision algorithm family while the later provides higher collision resistance with the penalty of lower performance. Other algorithms, such as MD5 or SHA1, will yield performance and collision resistance results between these bounds. Moreover, we measure the impact of each algorithm to the system's overall performance and ability to identify such attacks by covering a reasonable amount of kernel memory regions.

4.1 Experimental Setup

The system used for the evaluation of our kernel integrity monitor consists of an SGX-enabled Intel Core i7-6700 CPU, clocked at 3.40GHz, and 16GiB of DDR4 memory, clocked at 2400 MHz. The system is running Arch Linux with kernel version 4.14 and the latest version of the SGX software stack.

4.2 Snapshot Frequency

We begin the evaluation of our kernel integrity monitor by identifying the achievable snapshot frequency of each hashing algorithm in respect to the number of kernel memory regions provided. In this set of experiments, the system is instructed to monitor 8-byte long kernel memory regions, obtained using the `/proc/kallsyms` interface. Setting the system to its default mode of operation, that is using CRC-32, we perform constant monitoring of the provided memory regions while measuring the sustained snapshot frequency. We conduct the same experiment, each time increasing the number of monitoring addresses by 2,000, starting with 1,000 entries up to 20,000. The outcome of this experiment is depicted in Figure 3. As expected, we notice that as the number of memory locations increases, the frequency of monitoring the locations decreases linearly. Overall, the system is able to achieve a frequency of 42 KHz when monitoring 1,000 distinct kernel memory regions and provides monitoring frequencies greater than 10 KHz for configurations containing up to 4,000 memory locations.

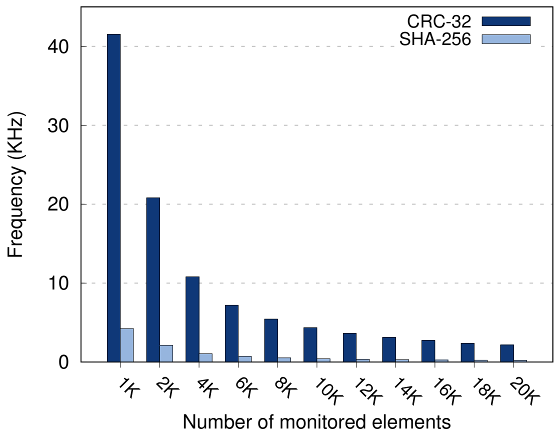

Figure 3: Sustainable scanning frequency using CRC-32 and SHA-256 in respect to the number of monitored regions.

We also perform the same set of experiments, this time by tuning the integrity monitor into using the SHA-256 algorithm for the checksumming operations. The results of this experiment are also shown in Figure 3. We observe that SHA-256, being more intensive algorithm compared to CRC-32, yields lower sustained monitoring frequencies. On average, its performance is one order of magnitude lower than CRC-32 and the system is able to achieve a scanning frequency of 4.3 KHz when monitoring one thousand distinct kernel memory regions.

4.3 Monitoring Accuracy

After obtaining the performance characteristics of our system using the two available hashing algorithms in multiple configurations, we want to determine its ability to accurately identify kernel side attacks. We base this analysis on the monitor's ability to detect the presence of a malicious self-hiding kernel loadable module. Similar case studies have also been conducted for the evaluation of kernel integrity monitoring systems found in the literature [9, 11].

In order to perform this measurement, we develop a custom self-hiding loadable kernel module (LKM) that performs zero malicious operations other than deleting itself from the loadable kernel modules list. With this setup we are able to stress our kernel integrity monitor to the maximum extend since this artificial module does not execute any code useful to an attacker, thus being exposed to the system for the minimum possible extend.

In typical Linux environments, loadable kernel modules are handled using the various available utilities such as `insmod` and

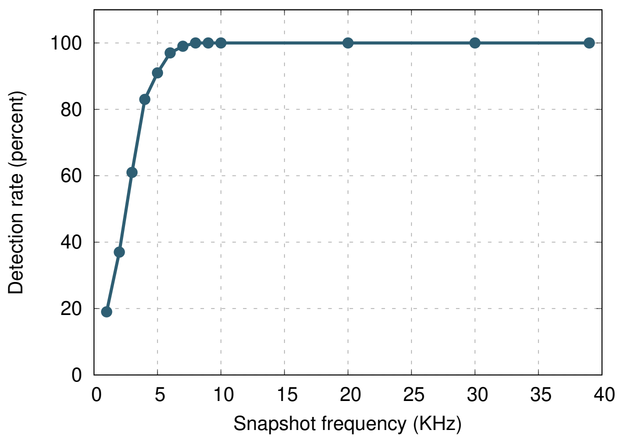

Figure 4: Detection rate of the self-hiding LKM with different snapshot frequencies. Frequency configurations of 8KHz or more achieve a 100% detection rate. For each frequency configuration we monitor the head of the kernel modules list and load a module that deletes its entry from the list 100 times.

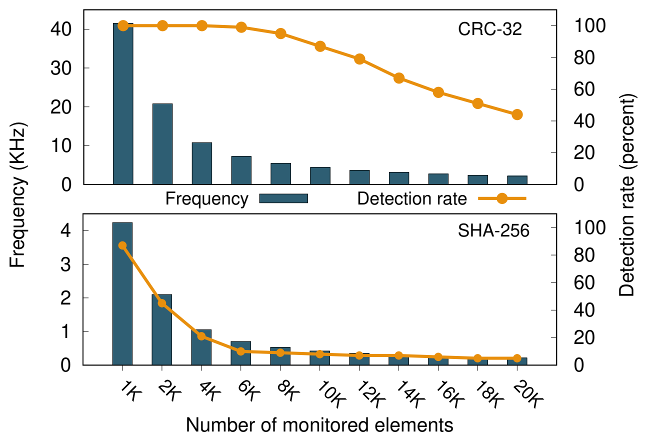

Figure 5: Scanning frequency and detection rate of the self-hiding LKM using CRC-32 and SHA-256 in respect to the number of monitored regions. The CRC-32 algorithm is faster and covers a larger memory space while achieving 100% detection rate. SHA-256 is able to cover a small memory region but provides higher security.

`rmmod`. These utilities are responsible for loading the modules into memory and invoking the appropriate system call for their initialization. The reverse operation is performed when a module needs to be unloaded from the system. During the initialization phase, the `insmod` tool opens the module object file and invokes the `finit_module` system call. This operation adds a handle to the kernel's loadable module list data structure, initializes any parameters given to the module and invokes the module's `init()` function. All modules currently present to the system can be obtained by iterating through this list. Once our artificial module is loaded, it hides itself from the system by removing its entry from the kernel's loadable module list. This transient malicious operation can be detected by our kernel integrity monitor by scanning the head of the loadable kernel modules list with a sufficient frequency.

With the following experiment we determine the minimum required monitoring frequency for the successful detection of such modifications of the kernel's data structure. We measure the detection rate using multiple frequency configurations while loading the self-hiding module 100 times. The outcome of this experiment is displayed in Figure 4. We observe that our monitor is able to reliably detect the presence of the malicious module with 100% accuracy when operating with a snapshot frequency of 8 KHz or more.

We conclude our evaluation by performing the first set of experiments, this time including the head of the kernel's loadable module list in each kernel memory region configuration while at the same time loading the self-hiding module into the system. Figure 5 presents the comparison of the sustained detection rate when monitoring with CRC-32 and SHA-256, using an increasing number of kernel memory locations. As We can see in the Y2 axis, our system is able to achieve 100% detection rate while scanning up to 6,000 distinct memory locations by using CRC-32 for the checksumming operations. On the other hand, when SHA-256 is used, the system achieves a maximum of 87% detection rate while scanning 1,000 memory locations. The increased level of security provided by SHA-256 comes with a substantial penalty to the system's overall performance. Nonetheless, for our specific hardware, SHA-256 can be used to successfully monitor up to 850 distinct memory locations of significant security importance.

5 RELATED WORK

Kernel integrity monitors can be divided into two main categories, (i) software based and (ii) hardware based, with the former often based on a hypervisor. Azab et al. [3] propose a system, similar to our design, that leverages ARM TrustZone [2] in order to perform kernel monitoring. The benefit of this approach is that the mappings and the virtual address translations of the kernel are transferred directly inside the trusted application, rendering the normal world unable to view or modify them. However, this model is only applicable to TrustZone capable devices such as IoT and mobile phones. HyperCheck [16] is a hardware-assisted tempering detection framework that leverages the CPU System Management Mode (SMM) and aims to protect the integrity of VMMs and the host's OS against certain classes of attacks. In contrast to SGX-Mon, where the analysis is performed in the secure enclave, executed on the target system, HyperCheck transmits the entire system state to an external server. Feng et al. introduce BehaviorKI [7], a behavior-triggered integrity checking system that extracts a set of behavior patterns by analyzing attacking processes and triggers checking with kernel invariants. OCsk [8] is a virtual machine based approach able to detect rootkits by determining violations to operating system invariants. Also, SecVisor [15] is a hypervisor based approach that protects the kernel's integrity by verifying that only user-approved code can execute in kernel mode thus protecting the OS against code injection attacks and rootkits. TrustAV [6], is an anti-malware monitor that leverages SGX in a similar way as SGX-Mon but targets executables and user files.

Copilot [13] is a snapshot based kernel integrity monitor that, similar to SGX-Mon, requires no modifications to the monitored system but is implemented on a custom FPGA. Moon et al. propose Vigilare [12], a hardware- and snooping-based integrity monitor that scans the bus for memory accesses that can possibly affect the kernel's security. Also, KI-Mon [11] extends Vigilare by offering event-triggered protection for mutable objects. Hypernel [10] is a security framework that combines a software and a hardware component, enabling a word-granularity monitoring capability on the kernel memory. Finally, Koromilas et al. propose GRIM [9], an external kernel integrity monitor that performs snapshot-based scanning by utilizing an external GPU.

6 DISCUSSION AND LIMITATIONS

Kernel Memory Mapping Manipulation. An attacker could manipulate the kernel memory mappings, defined by page tables, in order to fool the SGX-enabled integrity monitor and bypass detection. The attacker could manipulate the CR3 register and map the kernel code to a different physical page. In this work we assume that the kernel runs in a known state and is not compromised prior to the secure-bootstrap of the monitor, thus by monitoring the CR3 register such an attack is not feasible.

Denial-of-Service Attacks. Adversaries who manage to bypass the integrity monitor and compromise the system or malicious users able to acquire root access can easily disrupt the operation of Intel SGX. For example, they can kill or suspend the execution of enclaves. However, such activates can be easily detected using a custom heartbeat signal sent by SGX-Mon's enclave, utilizing random used-only-once beacons that can not be replicated or predicted.

SGX Side-Channel Attacks. Preventing potential side channel attacks is up to the enclave developer. The untrusted part of SGX-Mon's code base, residing out of the enclave, as well as the secure enclave must both be free of memory bugs, memory leaks, code vulnerabilities and error prone code. Additionally, the trusted code footprint must be small to limit attack surface. Moreover, side channel attacks targeting other parts of the software stack, hardware bugs or timing attacks are not taken into consideration in this work as they are orthogonal to our system. However, every work that aims to protect SGX-enabled software or hardware is applicable to our system and offers a direct benefit.

7 CONCLUSIONS

In this paper, we propose SGX-Mon, an external snapshot-based integrity monitor able to identify kernel side attacks that tamper and modify critical Linux kernel memory regions and data structures. Our system is executed in the user space, without any need for external hardware or kernel modifications and resides inside a secure Intel SGX enclave. Our system is able to remain untampered and undetected by malicious third parties and operates with a minimal TCB. SGX-Mon is able to achieve up to 100% accuracy while scanning up to 6,000 distinct memory locations and up to 87% accuracy while scanning up to 1,000 memory locations.

In the future, we aim to explore the effectiveness of random intervals in order to increase the system's overall accuracy when scanning more memory locations using highly collision resistant algorithms, such as SHA-256. Moreover, we plan to extend our system with a rule engine able to perform specific user-defined actions when a threat is detected as well as identify complex attacks targeting kernel code and registers.

8 ACKNOWLEDGMENTS

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. No. 2767). This work was also supported by the projects CONCORDIA, C4IIoT and COLLABS, funded by the European Commission under Grant Agreements No. 830927, No. 833828, and No. 871518. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

REFERENCES

[1] [n.d.]. Intel Software Guard Extensions. https://software.intel.com/en-us/sgx.

[2] ARM LIMITED. 2009. ARM Security Technology - Building a Secure System using TrustZone Technology.

[3] Ahmed M Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. 2014. Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 90–102.

[4] Yingxin Cheng, Xiao Fu, Xiaojiang Du, Bin Luo, and Mohsen Guizani. 2016. A Lightweight Live Memory Forensic Approach Based on Hardware Virtualization. *Information Sciences* 379 (07 2016). https://doi.org/10.1016/j.ins.2016.07.019

[5] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[6] Dimitris Deyannis, Eva Papadogiannaki, Giorgos Kalivianakis, Giorgos Vasiliadis, and Sotiris Ioannidis. 2020. TrustAV: Practical and Privacy Preserving Malware Analysis in the Cloud. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*. 39–48.

[7] Xinyue Feng, Qiusong Yang, Lin Shi, and Qing Wang. 2018. BehaviorKI: Behavior Pattern Based Runtime Integrity Checking for Operating System Kernel. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*.

[8] Owen S Hofmann, Alan M Dunn, Sangman Kim, Indrajit Roy, and Emmett Witchel. 2011. Ensuring operating system kernel integrity with OSck. *ACM SIGARCH Computer Architecture News* 39, 1 (2011), 279–290.

[9] Lazaros Koromilas, Giorgos Vasiliadis, Elias Athanasopoulos, and Sotiris Ioannidis. 2016. GRIM: leveraging GPUs for kernel integrity monitoring. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 3–23.

[10] D. Kwon, K. Oh, J. Park, S. Yang, Y. Cho, B. B. Kang, and Y. Paek. 2018. Hypernel: A Hardware-Assisted Framework for Kernel Protection without Nested Paging. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6.

[11] Hojoon Lee, Hyungon Moon, Daehee Jang, Kihwan Kim, Jihoon Lee, Yunheung Paek, and Brent ByungHoon Kang. 2013. Ki-mon: A hardware-assisted event-triggered monitoring platform for mutable kernel object. In *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*. 511–526.

[12] Hyungon Moon, Hojoon Lee, Jihoon Lee, Kihwan Kim, Yunheung Paek, and Brent Byunghoon Kang. 2012. Vigilare: toward snoop-based kernel integrity monitor. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 28–37.

[13] Nick L Petroni Jr, Timothy Fraser, Jesus Molina, and William A Arbaugh. 2004. Copilot-a Coprocessor-based Kernel Runtime Integrity Monitor.. In *USENIX security symposium*. San Diego, USA, 179–194.

[14] J. Rhee, R. Riley, D. Xu, and X. Jiang. 2009. Defeating Dynamic Data Kernel Rootkit Attacks via VMM-Based Guest-Transparent Monitoring. In *2009 International Conference on Availability, Reliability and Security*. 74–81.

[15] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. 2007. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. 335–350.

[16] Jiang Wang, Angelos Stavrou, and Anup Ghosh. 2010. Hypercheck: A hardware-assisted integrity monitor. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 158–177.

[17] Xiaoguang Wang, Yue Chen, Zhi Wang, Yong Qi, and Yajin Zhou. 2015. SecPod: A Framework for Virtualization-based Security Systems.