# Hardware-assisted Intrusion Detection by Preserving Reference Information Integrity

Junghee Lee[1], Chrysostomos Nicopoulos[2], Gi Hwan Oh[3], Sang-Won Lee[3], and Jongman Kim[1]

[1] Georgia Institute of Technology, Atlanta, USA
`junghee.lee@gatech.edu, jkim@ece.gatech.edu`
[2] University of Cyprus, Nicosia, Cyprus
`nicopoulos@ucy.ac.cy`
[3] Sungkyunkwan University, Suwon, South Korea
`wurikiji@skku.edu, swlee@skku.edu`

**Abstract.** Malware detectors and integrity checkers detect malicious activities by comparing against reference data. To ensure their trustworthy operation, it is crucial to protect the reference data from unauthorized modification. This paper proposes the Soteria Security Card (SSC), an append-only storage. To the best of our knowledge, this work is the first to introduce the concept of an append-only storage and its application to information security. The SSC framework allows only read and append operations, and forbids over-write and erase operations. By exploiting this trait, we can protect the reference data that must be updated constantly. It is demonstrated how SSC facilitates log protection and file integrity checking.

**Keywords:** log, hardware, protection, security

## 1 Introduction

There are numerous software tools that prevent or detect malicious activities in a computer system by comparing against reference data. For example, antivirus programs detect malware by matching files or snapshots of memory against signature databases. A signature database contains measurable characteristics of known malware. On the other hand, integrity checkers validate their target by comparing against the target's integrity information. For example, file integrity checkers validate files by checking if their checksum matches an a priori measured one. Additionally, log information – which is a trace of various server activities – also constitutes very important reference data when an attack is investigated. Thus, protecting reference data is critical for the above-mentioned techniques, because they would become useless once the reference data is contaminated.

Since reference data is constantly updated, we cannot protect it by enforcing a read-only property on it. Encryption can effectively protect data, as long as the key is not revealed. However, according to a recent report [1], 76% of data breaches occurring in 2012 exploited weak or stolen credentials. It is not a trivial task to keep the key itself secret. Using a virtual machine is also an effective way to protect or monitor important reference data. The host operating system (OS) can serve as the protector of reference data in the guest OSes. In a virtual machine environment, the integrity of the host OS becomes even more critical, because all the guest OSes are threatened once the host OS is compromised. There should be a mechanism to protect the reference data of the *host* OS.

Toward this end, this paper proposes the *Soteria*[4] *Security Card*, or SSC. SSC is a card that attaches to the host machine through a standard interface, such as Serial AT Attachment (SATA). The purpose of SSC is to protect reference data from unauthorized modification. SSC allows only read and append operations. Overwriting and erasing stored data is physically impossible. Therefore, it can be a secure foundation for storing important reference data. To the best of our knowledge, this work is the first attempt to introduce the concept of append-only storage and to apply it to information security. Note that write-once-read-many (WORM) devices can also be considered as append-only storage; compact disks (CD) and digital versatile disks (DVD) are typical examples. However, these devices are significantly slower than hard-disk drives and their media should be replaced once they become full. Instead, SSC offers better performance than WORM devices and has the necessary intelligence to sweep old data. Log protection and file integrity checking will be presented as two proof-of-concept examples of the capabilities of SSC.

The rest of this paper is organized as follows: the Soteria Security Card is presented in Section 2, followed by the two case-study examples of log protection and file integrity checking in Sections 3 and 4, respectively. Section 5 discusses related work, while Section 6 concludes this paper.

## 2 The Soteria Security Card: An Append-Only Storage Solution

To implement append-only storage, we have developed specialized hardware in the form of (a) an add-on card that can be attached to the host machine, (b) firmware running on the hardware, and (c) a device driver that provides an interface to the host OS. The architecture of the Soteria Security Card is explained in sub-section 2.1, followed by its implementation and performance evaluation results in sub-section 2.2.

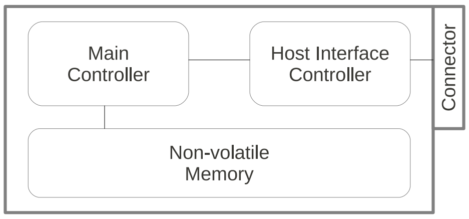

**Fig. 1.** The high-level block diagram of the developed *Soteria Security Card*.

### 2.1 The SSC Architecture

The *Soteria Security Card (SSC)* consists of an interface controller to the host system, non-volatile memory, and a main controller. A high-level block diagram of the SSC is illustrated in Figure 1.

[4] In ancient Greek mythology, Soteria was the goddess of safety and recovery from harm.

The interface controller to the host system connects the SSC to the host system through a standard system bus, such as Peripheral Component Interconnect (PCI), Serial AT Attachment (SATA), and Small Computer System Interface (SCSI). In our specific implementation, the SATA interface is employed. The interface controller handles the protocol of the system bus. The non-volatile memory is used to store the log files. The memory could be of any non-volatile type, such as NAND flash, NOR flash, Phase-Change Memory (PCM), or Spin-Transfer Torque Memory (STT-RAM). In this implementation, we adopted NAND flash memory. The main controller is an embedded processor where the firmware runs. An ARM7 microprocessor is used in the current implementation.

Figure 2 depicts the architecture of the firmware running on the SSC. The layers shown in the grey box are part of the firmware. The heart of the firmware is the file management layer. It manages files according to commands issued through the host interface. Through the host interface, the firmware accepts only read and append commands. The file management layer is a simple file system. It maintains a file descriptor table that keeps meta data for files, such as file name, file size, and position pointer. It maintains a main data region that stores data. Whenever an append command is issued, it appends data to the end of the data region and updates the meta data accordingly.

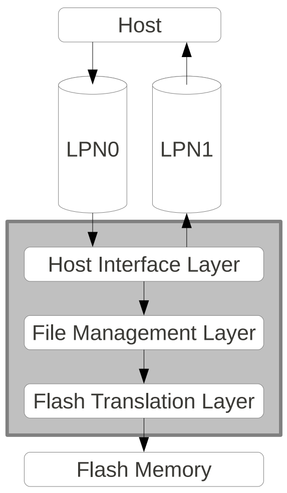

**Fig. 2.** The firmware architecture of the *Soteria Security Card*. The layers in the grey box are part of the firmware. The term "LPN" stands for Logical Page Number.

Since the capacity of NAND flash memory is limited, the data region eventually becomes full. The file management layer then deletes the oldest data to make space for new data. This is the only situation where the stored data are deleted. If this attribute is known to attackers, it could be exploited. However, various defense mechanism are possible, which depend on the application. Details of some defense mechanisms will be presented through the examples in the following sections.

The file management layer operates based on commands. The commands are given through the host interface. The host interface interprets the SATA protocol. One way to implement the commands to the file management layer is to modify the SATA protocol, because the SATA protocol also operates based on commands. For example, we may extend the SATA protocol using unused SATA commands. However, this requires modifications to the SATA device driver in the host OS. To minimize modifications, we implement the commands on top of the SATA protocol.

We introduce two logical pipes on top of the SATA protocol, as illustrated in the upper half of Figure 2. Logical page number 0 (LPN0) is always written by the host and LPN1 is written by the SSC. Under normal SATA protocol, writing to LPN0 means writing data to a physical page associated with LPN0. In SSC, this is interpreted as a command. Similarly, writing data to LPN1 is interpreted by the SSC as a response. From the perspective of the host, it is not required to modify the SATA device driver. Since there are many variations of SATA device drivers, depending on the manufacturer, modifying the entire SATA device driver is impractical. Instead, using two logical pipes requires only a simple device driver for the SSC. The SSC driver is, thus, independent of the manufacturer and platform, because it works over existing SATA device drivers.

### 2.2 Hardware Implementation

We have developed a prototype of the SSC framework by modifying the OpenSSD platform [2]. This setup has a SATA 2.0 interface with NCQ support. Its microcontroller is the ARM7TDMI-S running at 87.5 MHz. It has 96 KB of SRAM, 64 MB of SDRAM, and 64 GB of NAND flash memory. Figure 3 shows the prototype OpenSSD board attached to a host machine through the SATA interface. The host machine has an Intel Core i3 processor running at 3.3 GHz and having 4 GB of DDR3 SDRAM. The OS installed is Ubuntu 12.04, with a Linux kernel version of 3.9.2. The measured throughput of SSC in this environment is 100.71 KB/s.

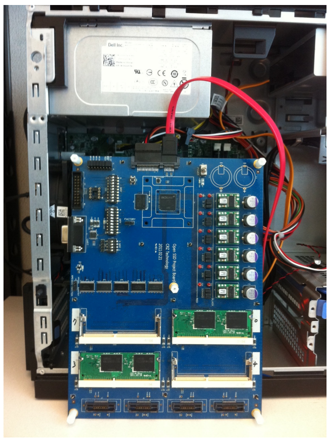
**Fig. 3.** The Soteria Security Card implemented on an OpenSSD board and attached to a host machine through the SATA interface.

## 3 Log Protection Through the Use of the SSC Framework

### 3.1 Problem Statement

The primary target of log protection is any server generating logs. When the administrator detects or suspects an attack, he/she usually investigates it by examining the logs. If the logs are contaminated or removed, it is extremely difficult for the administrator to deal with the attacks [3–6]. Even worse, the administrator may not even be aware of the attack(s) if the logs are fabricated. If the attacker manages to obtain root privileges, they can make changes to every file, including the logs.

This paper assumes the strongest adversary, i.e., one that can obtain root privileges and make changes to anything they want, including the OS, device drivers, file systems, and applications. Our goal is to prevent the attacker from modifying or removing the logs, even if they obtain root privileges. The ultimate goal of the proposed mechanism is to prevent an attacker from modifying *existing* logs. Of course, if the intruder obtains root privileges, they can stop logging, or they can start generating forged logs *after* the intrusion. However, they cannot make changes to *existing* logs that have been generated *before* the intrusion. If the attacker obtains root privileges by using some sort of hacking tool, the history of using the tool will be recorded in the logs and the logs would not be modifiable. Additionally, the location where the attack originated from will also be traced in the log file. Therefore, the logs would still hold valuable information for the administrator to investigate, even if the attacker obtains all-encompassing root privileges.

In summary, the scope of the proposed technique is as follows:

– **Target system:** Servers generating logs.
– **Threat model:** Attackers may have root privileges to modify or remove logs.
– **Goal:** Prevent attackers from modifying and removing existing logs.

## 3.2 Employing SSC for Log Protection

Since SSC is an append-only storage solution, it is very well suited to applications that maintain log information, whereby new data is constantly added, but existing data is not supposed to be overwritten. Figure 4 illustrates how SSC may be used to protect logs.

We do not change the existing path to store logs. When a server needs to generate logs, it uses Application Programming Interfaces (APIs) provided by a file system, such as open, write, and close. The file system updates meta data and log data. These are actually stored on a hard drive through a block device driver.

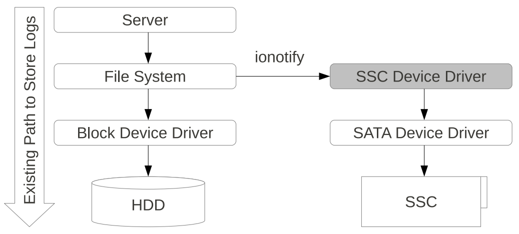

**Fig. 4.** High-level system overview of how SSC may be used to store and protect log information.

To snoop logs from the existing path, we exploit the `ionotify` capability that is supported by modern file systems. Users can be notified when an event occurs to any file. The SSC device driver can be notified when new log data is written to a log file. In the case of Linux, the shell command “`tail -F`” works using `ionotify`.

It should also be noted that the “`tail -F`” command does not incur significant performance overhead. This is because it does not keep monitoring (polling) the file, but, instead, the file system issues a notification when an event occurs. Also, this command runs in a separate process, independently from the server that generates logs. Since the write operation to the SSC is performed by a separate process – and not by the server itself – the access latency to the SSC does not affect the performance of the server.

The intruder may generate a huge amount of dummy logs, so that the old logs would be erased from the SSC memory. We call this a *log flooding attack*.

To cope with log flooding attacks, we first need to detect them. To do so, we employ a threshold-based mechanism. If the rate of incoming logs, in terms of events per second (eps), exceeds a pre-defined threshold for longer than a pre-defined period of time, a log flooding attack is declared. Since the range of normal logging rates varies with the system, the threshold should be determined by the administrator. When an attack is detected (or suspected), an alarm is sent to the administrator. Upon notification, the firmware intentionally delays any responses, so as to reduce the logging rate. If the responses from the SSC are slowed down, the storing of logs is delayed, which, in turn, slows down the generation of dummy logs. This reaction gives time to the administrator to find a way to handle the attack. The administrator may shut down the network, or copy existing protected logs to a safe place before valid logs are deleted. Each file is identified by its associated minor number. When SSC is installed, the administrator sets which minor number will store log files, so that the defense mechanism can be enabled only for the log files. After initialization is performed, the setting cannot be modified, unless SSC is physically re-installed.

### 3.3 Performance Evaluation

To evaluate the performance degradation incurred by the SSC, we measured the response time of the Apache 2.2.22 Web Server by using the Apache Benchmark [7]. When the SSC technique is adopted, logs generated from the web server – as well as those from the Linux OS – are recorded by the SSC. When SSC is not employed, all logs are stored as regular files on the hard drive. Table 1 summarizes the results.

The average response time (time per request) increases by a near-negligible 0.88%. In fact, the chosen settings correspond to a very pessimistic worst-case scenario for the SSC, because the response time is unrealistically small due to the very small size of the requested document (only 117 bytes). Moreover, we ran the Apache Benchmark on the same machine where the web server was running. In other words, the reported response time did not include any network delay. In real situations, the size of the request is usually much larger than 117 bytes, and requests are made remotely over the network. Thus, the absolute value of the response time is much larger than that of this experiment. In such situations, the overhead incurred by the SSC would be imperceptible.

**Table 1.** Comparison of Apache Web Server response times between an unprotected system and a system employing the SSC engine. Note that the average response time (time per request) increases by a near-negligible 0.88%.

| Item | No Log Protection | SSC |
|---|---|---|
| Document length | 117 bytes | 117 bytes |
| Number of requests | 10000 | 10000 |
| Time per request | 0.113 msec | 0.114 msec |
| | 100.00 % | 100.88 % |
| Transfer rate | 3.812 MB/sec | 3.794 MB/sec |
| | 100.00 % | 99.51 % |

The SSC framework aims to minimize its impact on the process(es) running on the server whose operations are being logged, by employing a separate process for storing the logs. Of course, the maximum logging rate supported by SSC is limited by the maximum attainable throughput of the card itself. The measured throughput of the SSC prototype used in this work is 100.71 KB/s. Assuming the log size of one event is 50 bytes, the SSC throughput can support up to 2062.54 events per second (eps). Obviously, the current SSC incarnation cannot be used in a server generating logs at a higher rate than this. However, according to a recent report [8], the average logging rate of various benchmarked devices is lower than 50 eps, and even the average *peak* rate is less than 2000 eps, except in one device (2414 eps). Therefore, even the throughput of the current prototype SSC is enough to support the logging needs of most modern servers. We envision the throughput of future SSC implementations to increase to levels well above the maximum peak logging rates of most (if not all) servers available on the market. The current implementation of the SSC has 64 GB NAND flash memory. Assuming 20 eps and 50 bytes per event, this capacity can accommodate 795 days (2.18 years) worth of logs. This is a sufficiently long time to retain logs, because 96% of data breaches are discovered within a few months [1].

## 4 File Integrity Checking by the SSC

### 4.1 Problem Statement

File modification is usually (if not always) a prerequisite or a result of malware. In order for malware to be installed, an existing file is modified, or a new file is secretly placed in the system. Some malware tries to hide itself by replacing existing software utilities, which results in file modification. Therefore, file integrity checking is a powerful tool to find out the cause of attacks and malware. The threat model is that the attacker may modify or install files. Further, our threat model assumes that the attacker may be able to contaminate the reference data.

The scope of the proposed technique is summarized as follows:

– **Target system:** Any type of server.
– **Threat model:** Attackers may modify or install files, and they may be able to update reference data in an authentic way.
– **Goal:** Detect file modification even if the reference data is contaminated.

### 4.2 Employing the SSC for File Integrity Checking

SSC is employed as storage for the reference data. The a priori measured integrity information is added to SSC in the following format: (*timestamp*, *comment*, *reference data*). The timestamp is the time the integrity information is measured. The timestamp is provided by the SSC. The comment is a short description that can be recognized by the administrator. The reference data is the actual integrity information.

Let us consider an example to illustrate how SSC works for integrity checking. The administrator wants to protect the `netstat` utility from unauthorized modification, right after a kernel upgrade. Its checksum is measured and stored to the

SSC with a timestamp “2013-09-12 09:11:11” and a comment saying “upgraded kernel to 3.10.11”. When `netstat` has not been modified further, the integrity checker displays a message “The current file is matched to version 2013-09-12 09:11:11” when the administrator runs the checker. If `netstat` has been modified by an attacker, but the reference data has *not* been breached, the current file cannot be matched with any record in the reference data. If `netstat` and the reference data have been modified, a new record is added to SSC with a different timestamp. Note that the timestamp is given by the SSC and *cannot be modified, nor erased under any circumstances.* If the timestamp and the comment are not recognizable to the administrator, the administrator can detect the file modification. Note that if the reference data is modifiable, the attacker may also modify the date of the data file, so that the data file may look unmodified. Detecting unauthorized modification of the timestamp can be automated. When the administrator runs the integrity checker, they specify a certain date when the last authorized modification was made. The integrity checker reports any file whose modification date is later than the given date, or whose integrity information does not match.

The performance of an integrity checker that employs the SSC is compared with an unprotected one in Table 2. The unprotected integrity checker stores reference data as a regular file on a hard-disk drive. Unlike the case of log protection, in this application the SSC affects the overall system performance, because a separate process is not employed. However, since the integrity checker is an off-line utility and runs in the background, the performance impact on foreground processes can be minimized by assigning a low priority.

**Table 2.** Comparison of response times between an unprotected integrity checker and an integrity checker employing SSC.

| Item | No protection | SSC |
|---|---|---|
| Number of files checked | 1000 | 1000 |
| Average file size | 108 KB | 108 KB |
| Checksum | MD5 | MD5 |
| Response time per file | 1.59 msec | 1.89 msec |

## 5 Related Work

Traditionally, hardware-assisted approaches for information security are involved in encryption (authentication) or monitoring.

The Trusted Platform Module (TPM) [9] is a hardware-assisted approach that offers secure generation of encryption keys. It can also encrypt data using the TPM endorsement key, which is burnt in the hardware during manufacturing. As mentioned in the introduction, once the key is revealed, or the software associated with the TPM is compromised, the encrypted data is not secure any longer. Instead of keeping keys, a security protocol using Physically Unclone-able Functions (PUFs) [10, 11] is an alternative way to provide authentication. PUF is a disordered physical system that cannot be reproduced exactly [10]. TrustZone [12], provided by ARM, enables the implementation of a secure execution environment, by separating the *secure world* from the *normal world.* The separation is facilitated by hardware. While the proposed SSC aims to protect data from unauthorized modification, the aforementioned approaches can protect data from both unauthorized modification and breaches. SSC can easily accommodate existing encryption techniques, if deemed necessary.

Another category of hardware-assisted security is monitoring. A separate hardware keeps monitoring the main system to check for attacks. This approach is especially useful for detecting rootkits, which compromise the OS kernel. Since rootkits reside within the kernel, it is hard to detect by software-only approaches. Copilot [13], RKRD [14], and KI-Mon [15] are typical examples.

Since logs contain essential information to detect and analyze intrusion, there have been many *software-based* techniques that provide log protection [3–6]. All of these techniques employ cryptography to prevent unauthorized access to logs. However, encryption-based approaches only make it harder to access logs, but not impossible. As described in Section 3, our threat model assumes that the attacker can obtain root privileges and make changes to any software component. In an extreme case, the attacker may delete everything on the hard drive in an unrecoverable manner. In this extreme case, the above-mentioned software-based techniques cannot protect logs from removal, even if the logs are encrypted. This problem still exists in the hardware-assisted log protection technique proposed by Boeck et al. [16], because the hardware is used only for authentication, but the log files are still stored on the hard disk.

File integrity checkers are usually part of Host-based Intrusion Detection Systems (HIDS). Zhang et. al [17] discuss the feasibility of implementing HIDS on a co-processor. In their implementation, the co-processor is attached to a Peripheral Component Interconnect (PCI) bus, whereby the device can issue commands to the main memory. The file integrity monitor can be secured by a virtual machine [18]. The monitor in the host OS checks the integrity of the guest OS.

## 6 Conclusion

Many software-based security solutions rely on reference data to prevent or detect malicious activities. It is crucial for such approaches to protect the reference data from unauthorized modification in order to guarantee their trustworthy operation. In this paper, we propose the Soteria Security Card, which is a hardware-based append-only storage solution for securing the reference data. SSC is a card attachable to the host machine through a standard bus interface. Since SSC allows only read and append operations, over-write and erase operations are physically impossible. Owing to its attributes, the SSC framework can facilitate efficient log protection and file integrity checking. Experimental results demonstrate that the performance degradation caused by the add-on hardware is negligible. The two example case studies presented in this paper (i.e., log protection and file integrity checking) merely serve as proof-of-concept. The SSC framework may be employed to protect any type of reference data.

## Acknowledgement

This research was partly supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2012R1A1A2A10044300) and the IT R&D program of MKE/KEIT [10041244, SmartTV 2.0 Software Platform].

## References

1. Verison: 2013 data breach investigations report (2013)
2. Chung, H.: Barefoot SSD controller technical reference manual (2011)
3. Takada, T., Koike, H.: NIGELOG: protecting logging information by hiding multiple backups in directories. In: Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on. (1999) 874–878
4. Waters, B., Waters, B.R., Balfanz, D., Balfanz, D., Durfee, G., Durfee, G., Smetters, D.K., Smetters, D.K.: Building an encrypted and searchable audit log. In: In The 11th Annual Network and Distributed System Security Symposium. (2004)
5. Schneier, B., Kelsey, J.: Secure audit logs to support computer forensics. ACM Trans. Inf. Syst. Secur. **2**(2) (May 1999) 159–176
6. Kawaguchi, N., Ueda, S., Obata, N., Miyaji, R., Kaneko, S., Shigeno, H., Okada, K.: A secure logging scheme for forensic computing. In: Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC. (2004) 386–393
7. Foundation, A.S.: Apache HTTP server
8. Butler, J.M.: Benchmarking security information event management (SIEM) (2009)
9. Group, T.C.: Trusted platform module (TPM) specifications (2011)
10. Ruhrmair, U., van Dijk, M.: Pufs in security protocols: Attack models and security evaluations. In: Security and Privacy (SP), 2013 IEEE Symposium on. (2013) 286–300
11. Brzuska, C., Fischlin, M., Schrder, H., Katzenbeisser, S.: Physically uncloneable functions in the universal composition framework. **6841** (2011) 51–70
12. ARM: ARM security technology (2009)
13. Petroni, Jr., N.L., Fraser, T., Molina, J., Arbaugh, W.A.: Copilot - a coprocessor-based kernel runtime integrity monitor. In: Proceedings of the 13th conference on USENIX Security Symposium - Volume 13. (2004)
14. Grover, S., Khosravi, H., Kolar, D., Moffat, S., Kounavis, M.: Rkrd: Runtime kernel rootkit detection. **48** (2009) 224–236
15. Lee, H., Moon, H., Jang, D., Kim, K., Lee, J., Paek, Y., ByungHoon, K.B.: KI-Mon: a hardware-assisted event-triggered monitoring platform for mutable kernel object. In: Proceedings of 22nd USENIX Security Symposium. (2013)
16. Boeck, B., Huemer, D., Tjoa, A.M.: Towards more trustable log files for digital forensics by means of “trusted computing”. In: Advanced Information Networking and Applications, 24th IEEE International Conference on. (2010) 1020–1027
17. Zhang, X., van Doorn, L., Jaeger, T., Perez, R., Sailer, R.: Secure coprocessor-based intrusion detection. In: Proceedings of the 10th workshop on ACM SIGOPS European workshop. EW 10, New York, NY, USA, ACM (2002) 239–242
18. Quynh, N.A., Takefuji, Y.: A novel approach for a file-system integrity monitor tool of xen virtual machine. In: Proceedings of the 2nd ACM symposium on Information, computer and communications security. (2007) 194–202