Intrusion Detection Against Unauthorized File Modification by Integrity Checking and Recovery with HW/SW Platforms Using Programmable System-On-Chip (SoC)

Mochamad Julianto S
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
muhamad.info@gmail.com

Rinaldi Munir
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
rinaldi.munir@itb.ac.id

Abstract— **The attacks of modifying files such as website hacking, virus infection and ransomware are becoming a recent issue. This is due to a lack of attention to the programs or maintenance of web applications after it has been completed and connected to the internet, while hackers will always try to find a security hole to infiltrate the system. The security of software-based system used in the market today is not good enough to protect those attacks because the software-based protection, in general, can still be modified or manipulated. Therefore, a mechanism that can protect files in a system (such as personal computer or server) by both software and hardware is required. Implementing the mechanism to a hardware can bring a better immunity from malware infections. This paper proposed a method that provides protection mechanism against unauthorized file modification using the existing Integrity Checking and Recovery (ICAR) concept by holistic approach (hardware and software protection) with an open source security-oriented platform using a programmable system on chip (SoC). The results of the simulations show that the system can protect the authenticity of files against file modification-based attacks in the limited scenarios of attack without modifying main system configuration.**

***Keyword---system on chip, integrity checking, malware, intrusion detection, web defacement*.**

I. INTRODUCTION

Nowadays, the development of information technology causes many efforts of hacking or corrupting a system (PC/server) by manipulating it, for example, an illegal modification of the system configuration or sending malware on specific systems to open security holes and commit crimes. The Indonesian government's incident response team said that the vulnerability of a system could be due to several things, such as the lack of protection mechanisms against files stored in the system storage, operating system vulnerabilities and program code vulnerabilities when building website pages. Once the vulnerabilities are known, then the hacker will exploit it to gain access to the system as root or administrator. When exploiting system vulnerabilities, attackers usually use malicious code or commonly referred as exploit code [7]. These actions are also called intrusions and most of them cause file modifications [4]. This can be a serious threat because a successful attack may lead to system failure.

There is a concept of digital checksum which calculates a unique value from a file content called an Integrity Checking and Recovery (ICAR) [4]. The conceptual approach is to prevent file modification of important data such as file systems in the operating system. This concept can be very powerful to prevent previously mentioned threats such as preventing malicious code to infect or modify important files (e.g., web server configuration, web pages, file system, etc.) stored in system storage. Also, it protects website pages by preventing web defacement and, at the same time, protecting all the crucial ICAR data including binaries, file backups and hash database inside write-protected storage.

ICAR was implemented as an in-kernel Linux Security Modules (LSM). Besides affecting system performance, modification of kernel level may also increase the attack surface. By enhancing this method using a different approach, it might be possible to integrate security system outside the physical system without modifying the main system (e.g., kernel, operating system configuration, create a rootkit, etc.) in existing infrastructure. This approach is expected to suppress the attack surface.

Faced with this challenge, holistic security approach that combines logical and physical aspect to provide better immunity from contamination, malicious code infections or vulnerability must be taken to protect files in the system [1]. The physical approach used in this study is an open source security-oriented hardware and software platform called system on chip (SoC). SoC is an Integrated Circuit (IC) that integrates all components of a computer or other electronic system into single chip [5]. SoC design allows high performance, good

978-1-5386-0954-5/18/$31.00 ©2018 IEEE

processing technology, miniaturization, efficient battery lifetime and cost sensitivities.

In this research, we propose a mechanism to overcome file modification attack to protect the integrity of crucial files stored in system storage using ICAR method with some modifications to the model layer. Furthermore, implementation using SoC module that is applied as a separate physical system can also monitor file integrity on specific files in the main system. The enhanced system can also perform valid updates in real time to checksums database and backup files through a security procedure such as authentication mechanism to ensure that the system performs updates by an authorized user only.

II. RELATED WORKS

Some of the threats to computer system security include virus, Trojan programs, rootkit, and others. Most computer system attacks are performed using network access and the likelihood of security breach increases if the computer system is not adequately protected [4]. So it needed necessary to introduce some methods to check whether there has been a breach or not which are done most effectively by checking for unauthorized file modification [3]. There are several concepts to overcome issues related to the threats.

In-kernel Integrity Checker and Intrusion Detection File System (I3FS) concept is an in-kernel system to detect intrusion through integrity checks. It compares the checksum of files in real-time. This system approach is capable of discovering any failure in integrity check and it immediately blocks access to the affected file and notifies the administrator. Moreover, I3FS is implemented inside the kernel as a loadable module. It assumes that the file system is the most appropriate location for security modules because most intrusions would cause file modification [4].

The advantage of this system is that it has an authentication mechanism. Therefore, only valid updates to the files that carry policies should be permitted through a secure channel as to prevents malicious programs from triggering checksum updates subsequent to an unauthorized modification to file data. This is due to critical programs and files need to be updated occasionally and such updates should not require re-initialization of the file system [9].

Another concept is called Integrity Checking and Recovery (ICAR) system. This system concept utilizes the mechanism of a unique digital fingerprint, also known as a checksum, is stored in a secured database and retrieved to check if an unauthorized modification to a file has been made. If the calculated hash differs from the initial hash, a security procedure is activated to restore the original file content from the backup and notify system administrator. The file integrity checking mechanism is implemented as kernel level security module which supplies higher security than user level protection. This mechanism was designed to protect file contents, especially considering the important configuration and file system [4].

The system provides the advantages of backup mechanism to ensure that the files are restored when file intrusion detected and it uses write-protected media to store crucial files of the security system such as cryptographic hash, file backup, and security binaries to eliminate the threat of unauthorized modification to them.

As mentioned earlier, this work adapted the ICAR system concept with an approach of methodology to implement holistic security on hardware and software system architecture with an open source security-oriented platforms using SoC device named Secube^{TM} development board [1].

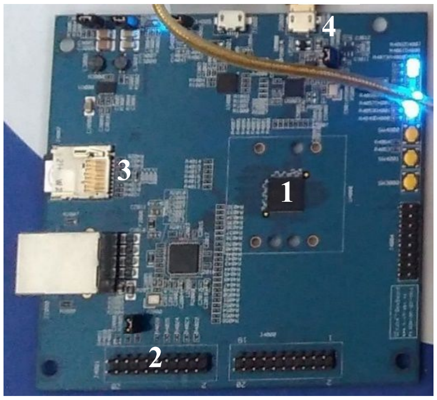


Fig. 1 Secube^{TM} development board

As shown in Fig. 1, the hardware components used in development system are the programmable chip (no.1) (provided with a JTAG interface (no.2) for programming, debug and testing operations), the external flash memory (no.3), and the physical USB interface (no.4) to support request services.

III. PROPOSED APPROACH

Currently, the proposed approach has two main APIs software libraries [1]. First, the device-side software includes all device functionalities implement on embedded CPU. Second, the host-side software includes software that can run on other physical entities such as PC or server. This host-side software is an adapter of the entities to request service from the device through device-side software and the physical communication from host to request services to the device are supported through USB interface.

III.1. System Design

The proposed system consists of three layers as shown in Fig. 2. System layers are as follows:

1. *Application Layer*: consists of two modules - First, the security module is responsible for the file integrity verification and verify the integrity of backup copies of protected files stored in device storage. Second, device adapter module is responsible to handle authentication procedure and request a service to the device.
2. *Hardware Layer*: contains device-side software provides functionalities that include login/logout, key management,

cryptography, and storage manager to manage database synchronization operations and write-protected manager as a controller for write permission to write-protected device storage.

3. *Data Layer:* consists of crucial data for the security mechanism such as security database that stores file hashes, metadata from the protected file and backup copies of protected files. These files are stored in a write-protected storage.

The write-protected storage in this proposed system has a flexible function managed by the write-protected manager that is included in the device-side software. This is used to control access permission to write into or modify the file inside device flash storage. Thus, only authorized actions for modifying files stored in the device storage.

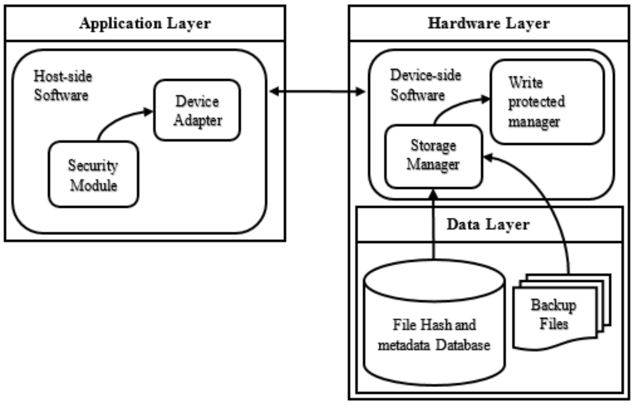


Fig. 2 Modified ICAR system layers

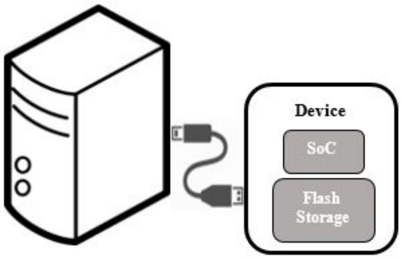


Fig. 3 Design implementation architecture

The application layer is an external device library that is designed to be scalable, they may portably run on a different operating system, thus limiting the usage of platform-dependent modules. Practically, they run on host OS layer to improve portability and migration, the libraries of host-side software are organized in device write-protected storage.

The hardware layer consists of device-side software that provides functionalities as mentioned earlier. The data layer is applied inside hardware layer. It is used to store copied files and security database in device storage. It is also protected by write-protected manager function in device-side software. If the security module in the application layer detects an unauthorized modification of protected files, it will automatically restore the original file content from the backup file in the data layer. Fig. 3 illustrates the system implementation architecture and shows the position of data layer on the hardware structure.

III.2. Protection Mechanism

The proposed system flow consists of two protection mechanisms that include backup and restores processes. The system also records any process information in system log to simplify administrators work to monitoring and evaluating the system.

III.2.1. Backup Mechanism

A whole backup process covers the activity of scanning files in a directory specified by an authorized user and all process including authentication, backup, and logs records. The authentication mechanism consists of two-level processes. The first one is a legitimate process for receiving a key from device key manager used to decrypt password string from user input. The second one is an authentication process that uses the password to gain write access permission to device flash memory, it deactivates write-protected function during a write process. After the writing process is complete, the write-protected function is reactivated.

Fig. 4 shows the algorithm for the backup process used by the proposed system. This process can run parallel with the file monitoring process as it covers functions including selection of directory where files will be protected, generating cryptographic hashes, extracting file information (i.e. name, path, size, last modified date) to store in security database and files backup which are automatically stored in the data layer.

III.2.2. Restore Mechanism

The host system protection algorithm detail of integrity verification and recovery process is shown in Fig. 4. It is responsible to create a list of files in the selected directory, generate hashes, and extract file information (i.e. name, path, size, last modified date) before requesting services from the device to compare each file detail to the security database. It is also responsible for generating and executing a command to request services from the device through device adapter. Fig. 5 shows the algorithm to check file information in the security database. If there is an information about the current file, it is assumed that the file should not be there (it could be a virus/worm or Trojans) then the system will remove it. If there is an information about the file, the system will calculate a hash and extract meta information (i.e. name, path, size, last modified date) from the file and then compare them to security database.

If they are not the same, the system will remove the modified file and then activate recovery procedure to restore the original file content.

Additionally, there are two verification processes before and after restoring file content procedure executed to ensure restored files are correct. Before the original file content restored, the system will ensure whether the backup file is clean or corrupted by comparing the hash of backup file with the security database. If the comparison value is different, it is assumed that the backup file is corrupted, then it will automatically remove from the system. Furthermore, after the restore process is completed, the system verifies the restored file whether it is correct or not by comparing hashes value with the security database. If the restored file is incorrect, the system will repeat the attempt of restoring the corrupted file.

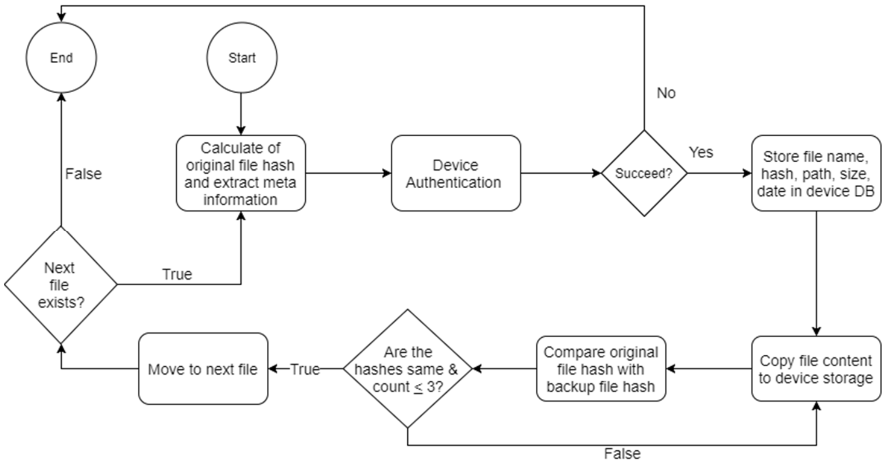

End
Start
No
Calculate of original file hash and extract meta information
Device Authentication
Succeed?
Yes
Store file name, hash, path, size, date in device DB
False
Next file exists?
True
Move to next file
True
Are the hashes same & count ≤ 3?
Compare original file hash with backup file hash
Copy file content to device storage
False

Fig. 4 Backup process algorithm

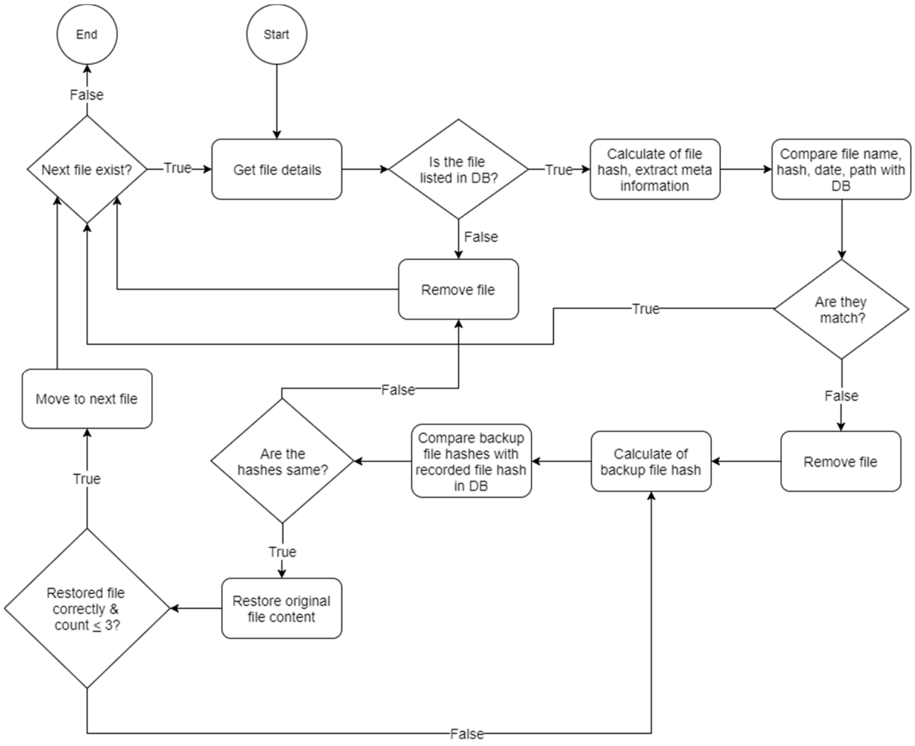

End
Start
False
Next file exist?
True
Get file details
Is the file listed in DB?
True
Calculate of file hash, extract meta information
Compare file name, hash, date, path with DB
False
Remove file
True
Are they match?
Move to next file
False
False
Are the hashes same?
Compare backup file hashes with recorded file hash in DB
Calculate of backup file hash
Remove file
True
True
Restored file correctly & count ≤ 3?
Restore original file content
False

Fig. 5 Recovery process algorithm

IV. IMPLEMENTATION

The proposed system was implemented in C language and it included device-side (chip firmware) and host-side software. The device-side software was injected into the programmable processor so that the device system can perform its function to provide services. The host-side software (device adapter) are run as user-level tool, it is installed in the device write-protected flash memory.

Fig.6 shows an implementation of the system. As is shown, the hardware architecture is implemented as an external device. This approach isolates device functionality separated from the main system. Currently, to support portability, there are no modifications at the operating system level including kernel or any file systems and there are no user application installation processes required. The physical communication between the device and main system are done via a USB interface.

In this paper, we define scenarios for intrusion detection using simulation such as malware (i.e. virus/worm and Trojan), unauthorized file modification and web defacement.

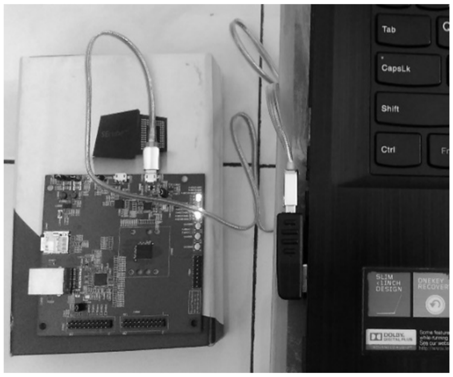
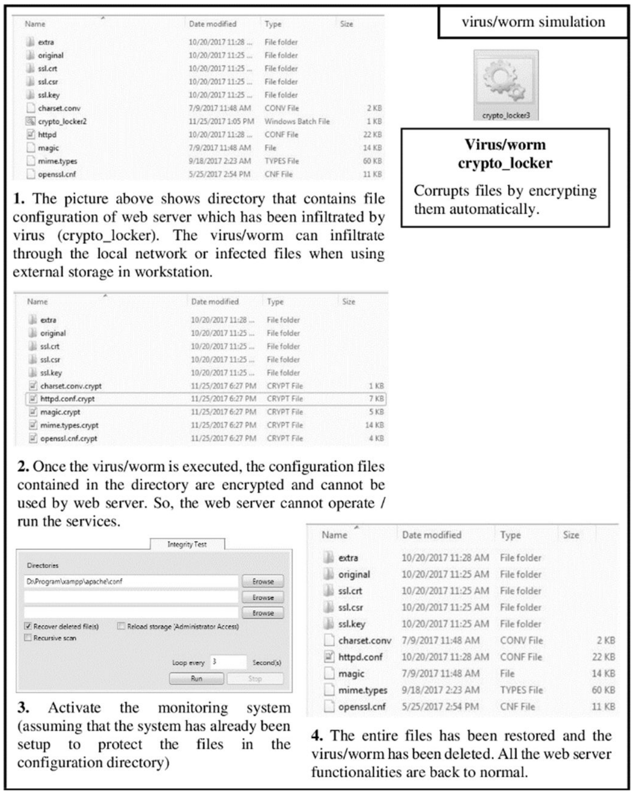


Fig. 6 Implementation for simulation

V. EXPERIMENT RESULTS AND EVALUATION

Based on the design in Fig. 6, we test the response of security mechanism based on the scenario that already mentioned in section IV. The goal of this simulation is to know the advantages of using hardware and software protection.



virus/worm simulation

Name	Date modified	Type	Size
extra	10/20/2017 11:28 ...	File folder	
original	10/20/2017 11:25 ...	File folder	
ssl.crt	10/20/2017 11:25 ...	File folder	
ssl.csr	10/20/2017 11:25 ...	File folder	
ssl.key	10/20/2017 11:25 ...	File folder	
charset.conv	7/9/2017 11:48 AM	CONV File	2 KB
crypto_locker2	11/25/2017 1:05 PM	Windows Batch File	1 KB
httpd	10/20/2017 11:28 ...	CONF File	22 KB
magic	7/9/2017 11:48 AM	File	14 KB
mime.types	9/18/2017 2:23 AM	TYPES File	60 KB
openssl.cnf	5/25/2017 2:54 PM	CNF File	11 KB

Virus/worm crypto_locker

Corrupts files by encrypting them automatically.

1. The picture above shows directory that contains file configuration of web server which has been infiltrated by virus (crypto_locker). The virus/worm can infiltrate through the local network or infected files when using external storage in workstation.

Name	Date modified	Type	Size
extra	10/20/2017 11:28 ...	File folder	
original	10/20/2017 11:25 ...	File folder	
ssl.crt	10/20/2017 11:25 ...	File folder	
ssl.csr	10/20/2017 11:25 ...	File folder	
ssl.key	10/20/2017 11:25 ...	File folder	
charset.conv.crypt	11/25/2017 6:27 PM	CRYPT File	1 KB
httpd.conf.crypt	11/25/2017 6:27 PM	CRYPT File	7 KB
magic.crypt	11/25/2017 6:27 PM	CRYPT File	5 KB
mime.types.crypt	11/25/2017 6:27 PM	CRYPT File	14 KB
openssl.cnf.crypt	11/25/2017 6:27 PM	CRYPT File	4 KB

2. Once the virus/worm is executed, the configuration files contained in the directory are encrypted and cannot be used by web server. So, the web server cannot operate / run the services.

Integrity Test

Directories

D:\Program\xampp\apache\conf [Browse]

[Browse]

[Browse]

☑ Recover deleted file(s) ☐ Reload storage (Administrator Access)

☐ Recursive scan

Loop every 3 Second(s)

[Run] [Stop]

3. Activate the monitoring system (assuming that the system has already been setup to protect the files in the configuration directory)

Name	Date modified	Type	Size
extra	10/20/2017 11:28 AM	File folder	
original	10/20/2017 11:25 AM	File folder	
ssl.crt	10/20/2017 11:25 AM	File folder	
ssl.csr	10/20/2017 11:25 AM	File folder	
ssl.key	10/20/2017 11:25 AM	File folder	
charset.conv	7/9/2017 11:48 AM	CONV File	2 KB
httpd.conf	10/20/2017 11:28 AM	CONF File	22 KB
magic	7/9/2017 11:48 AM	File	14 KB
mime.types	9/18/2017 2:23 AM	TYPES File	60 KB
openssl.cnf	5/25/2017 2:54 PM	CNF File	11 KB

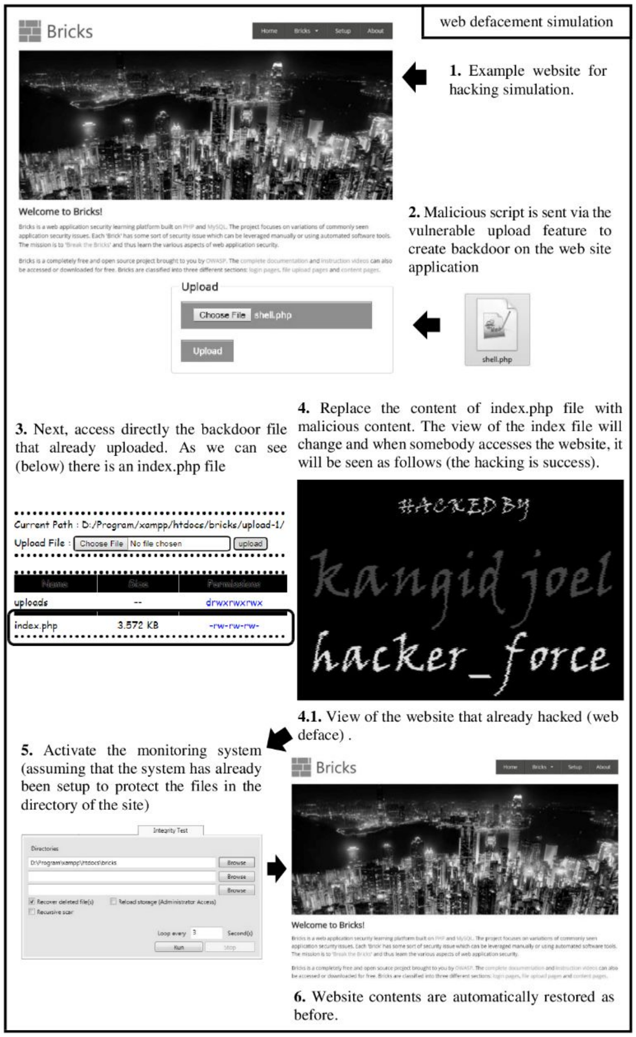
4. The entire files has been restored and the virus/worm has been deleted. All the web server functionalities are back to normal.

Fig. 7 Example of virus/worm attack scenario

Before beginning all simulations scenario, first of all, we specified directories including their files to be protected by doing the backup process on them and started the monitoring process. In this experiment, the files included configuration of a web server and web site pages. We use them to simulate unauthorized file change made by virus/worm infection or manually by unauthorized users (i.e. hackers).

Fig. 7 shows an example of malware attack scenario, we use virus/worm which has the ability to encrypt files and Trojan that carry the virus/worm to simulate infection or modification to them. When an infection occurs, the restore procedure is triggered and it automatically remove infected files including illegal files inside protected directories.

In website defacement example scenario (Fig. 8), we simulate attacks using a malicious script to create a backdoor to modify pages by inserting malicious code into it and also modifying configuration file including website and web server setting. The recovery procedure also active when the unauthorized modification occurs on the protected files.



web defacement simulation

Welcome to Bricks!

Bricks is a web application security learning platform built on PHP and MySQL. The project focuses on variations of commonly seen application security issues. Each 'Brick' has some sort of security issue which can be leveraged manually or using automated software tools. The mission is to 'Break the Bricks' and thus learn the various aspects of web application security.

Bricks is a completely free and open source project brought to you by OWASP. The complete documentation and instruction videos can also be accessed or downloaded for free. Bricks are classified into three different sections: login pages, file upload pages and content pages.

1. Example website for hacking simulation.

Upload

[Choose File] shell.php

[Upload]

2. Malicious script is sent via the vulnerable upload feature to create backdoor on the web site application

3. Next, access directly the backdoor file that already uploaded. As we can see (below) there is an index.php file

Current Path : D:/Program/xampp/htdocs/bricks/upload-1/

Upload File : [Choose File] No file chosen [upload]

Name	Size	Permissions
uploads	--	drwxrwxrwx
index.php	3.572 KB	-rw-rw-rw-

4. Replace the content of index.php file with malicious content. The view of the index file will change and when somebody accesses the website, it will be seen as follows (the hacking is success).

4.1. View of the website that already hacked (web deface) .

5. Activate the monitoring system (assuming that the system has already been setup to protect the files in the directory of the site)

Integrity Test

Directories

D:\Program\xampp\htdocs\bricks [Browse]

[Browse]

[Browse]

☑ Recover deleted file(s) ☐ Reload storage (Administrator Access)

☐ Recursive scan

Loop every 3 Second(s)

[Run] [Stop]

Welcome to Bricks!

Bricks is a web application security learning platform built on PHP and MySQL. The project focuses on variations of commonly seen application security issues. Each 'Brick' has some sort of security issue which can be leveraged manually or using automated software tools. The mission is to 'Break the Bricks' and thus learn the various aspects of web application security.

Bricks is a completely free and open source project brought to you by OWASP. The complete documentation and instruction videos can also be accessed or downloaded for free. Bricks are classified into three different sections: login pages, file upload pages and content pages.

6. Website contents are automatically restored as before.

Fig. 8 Example of web defacement scenario

The summary of comparison between the conventional system (ICAR) with the proposed mechanism can be seen in the following table.

Table 1 Criteria comparison [6]

Comparison Criteria	**Conventional ICAR design**	**Proposed modified design**
Installation: How easy the installation can be done	The system is implemented as kernel extension, so modification of system kernel is required	No installation required
Installation Prerequisites: What the system needs before installation	Need OS kernel modification and patch the kernel with grsecurity module	Portable build system, no prerequisites module required
Attack surface: The possibility of increasing attack surface during system implementation	System implementation need modification on the OS kernel that would increase the chance of the attack surface during implementation	System implementation does not need to modify any OS file system or kernel
Automatic Background Job: System can automatically run as background job	Yes	Yes
Multiple Platform: System can be deployed in multi platform such as windows, unix base	No (in-kernel linux)	Yes
Protection of IDS Binaries: Are the binaries of the intrusion detection files protected?	Yes, the files are protected using write-protected external storage	Yes, the files are protected inside SoC device flash storage
Protection of Integrity Database: Is the database integrity protected	Yes, it is protected inside write-protected external storage	Yes, it is protected inside SoC device storage

VI. CONCLUSION AND FUTURE WORK

The proposed mechanism presented the solutions with a different approach which enable holistic security using hardware and software platforms. Comparing with the existing system, there are several advantages as follows:

1 *Hardware protection*: The security service is treated as an extension of the hardware layer and the functionalities are isolated make it more robust and have better immunity,

2 *Flexible write-protected storage*: the proposed design allows writing into device flash storage through valid authentication procedure to update some files when necessary, so there is no need to recreate initial backup using write-protected storage separate from protection operation every time authorized modifications are needed. The authentication procedure is handled by device-side software (chip firmware),

3 *Portable*: With this approach, there is no need to modify operating system in the main system. The major benefit of the solution is that the mechanism is separate from the protected system, which enables the implementation of this method in different platforms.

On the other hand, when the extended hardware that provides protection mechanism encounter failure (i.e. connection issue, hardware failure, etc.), there is no fail-safe operation that can respond to recovery procedure to the threat to the main system. So, the system should be made available to encounter failure to the issues mentioned earlier. Besides it also increases the cost of hardware.

REFERENCES

[1] Farulla, A. G., Prinetto, P., Varriale, A. (2017): Holistic Security via Complex HW/SW Platforms. *12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (OTIS).*

[2] Varriale, A., Vatajelu, E. I., Natale, G. D., Prinetto, P., Trotta, P., Margaria, T. (2016): SEcube™: An Open-Source Security Platform in a Single SoC. *International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS).*

[3] Adukkathayar, C. A., Krishnan, G. S., Sasikumar, G. (2015): Advanced integrity checking and recovery using write-protected storage for enhancing operating system security. *Computer Science & Education (ICCSE)*.

[4] Kaczmarek, J. dan Wrobel, M. R. (2014): Operating system security by integrity checking and recovery using write-protected storage. *IET Information Security*.

[5] Rajesvari, R., Manoj, G., Ponrani, M. (2013): System-on-Chip (SoC) for Telecommand System Design. *International Journal of Advanced Research in Computer and Communication Engineering*

[6] Grim, L. dan Vandenbrink, R. (2014): IDS: File Integrity Checking. *SANS Institute Infoces Reading Room.*

[7] Standard Operating Procedure Incident Handling Web Defacement, Retreived from: https://govcsirt.kominfo.go.id/download/SOP/SOP IH_Web_Defacement(2).pdf. 2017-Jul-22.

[8] Christodorescu, M., Jha, S., Maughan, D., Song, D., Wang, C. (2007): Malware Detection. Springer Science. **1,** X-XI.

[9] Patil, S., Kashyap, A., Sivathanu, G., Zadok, E. (2004): I3FS: An in-kernel integrity checker and intrusion detection file system. *Proceedings 18th USENIX Annual Large Installation System Administration Conference (LISA2004)*.

[10] McGraw, G., Morrisett, G. (2000): Attacking malicious code: report to the Infosec research council. *IEEE Software*, **5**, 33 - 41.