National College of Ireland

Optimizing FIM System Using YARA Rules

MSc Academic Internship
MSc Cyber Security

Kedar Sunil Wattamwar
Student ID: 2211653

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Kedar Sunil Wattamwar……

Student ID: ……22116532……

Programme: ……MSc Cyber Security…… **Year:** .2022-2023..

Module: ……MSc Academic Internship……

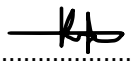
Supervisor: ……Mr. Vikas Sahani……

Submission Due Date: ……14-12-2023……

Project Title: ……Optimizing FIM System using YARA rules……

Word Count: ……7051…….. **Page Count**……19……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: ……14-12-2323……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimizing FIM Systems using YARA rules

Kedar Sunil Wattamwar
22116532

Abstract

In terms of functionality and confidentiality, sensitive files in computer systems, such as log files, executable programmes, configuration, and authorization data, are extremely important. By confirming every operation taken on these sensitive files, an efficient method known as file integrity monitoring is suggested to identify aggressive behaviours and safeguarding sensitive data. This paper presents a solution which continuously check the integrity of files and also gives an alert for addition or deletion of files. The method is also capable to detect and report if the added file is malicious or not. This research is significant because it has the potential to improve computer system security by lowering the possibility of malicious or unauthorised file additions or modifications, which lowers the chance of security breaches and system disruptions.

Key words: File integrity monitoring, Yara rules, SHA256, Malicious file input, file integrity.

1 Introduction

The topic of file protection has consistently gained attention from both academic and industrial sectors. In today's digital landscape, a vast repository of data and critical information is stored within these files, this makes it important to secure them. Unauthorised file access is a contributing factor in many security incidents, as seen by the WannaCry ransomware attack[1] In this kind of attack an executable file was inserted in the system, which created a backdoor program that was used to share sensitive data to the attacker's server. The attackers later changed the logs in the system, altering the integrity which left no traces to detect the attack sooner. The presented solution effectively detects these changes in the system by periodically checking the integrity of the files and directories in the system.

There are several critical factors that underscore the significance of file integrity monitoring in today's digital landscape, data is at the core of most activities, both in personal and professional contexts. Protecting this information from unauthorized access, tampering, or loss is crucial. There are many cases where there are significant consequences, such as data breaches, financial losses, and reputational damage[2].

Any unauthorized remote access in any organization system can allow them to execute command at OS. This kind of attacks also known as shell attacks work using typically small

[1] https://sbgsmedia.in/2018/05/10/2261f190e292ad93d6887198d7050dec.pdf
[2] https://www.csoonline.com/article/534628/the-biggest-data-breaches-of-the-21st-century.html

piece of malicious code written in various form of coding languages like java, python php, jsp, etc. These malicious codes are usually implanted in an executable file which when executed gives remote access to the command line of the system to attackers. The attack can be performed as bind shell or reverse shell. In bind shell the target machine acts as a listener, but traditional firewalls usually detect this attack and block them. However reverse shell, where the attacker machine is the listener, it can bypass these firewalls because this time the target machine tries to connect to the attacker machine, which is an inside out connection. To perform this kind of attack an initial access is required for an attacker to implant these malicious code files in the system, the presented solution not only checks the integrity of the file but also detects any addition of files. These files when recognized as added are then immediately scanned using a library of Yara rules which can notify the user with the details of the file been added. If the file is detected as malicious, further actions can be performed to isolate the machine and blacklist the Internet protocol (IP) addresses connected to the system.

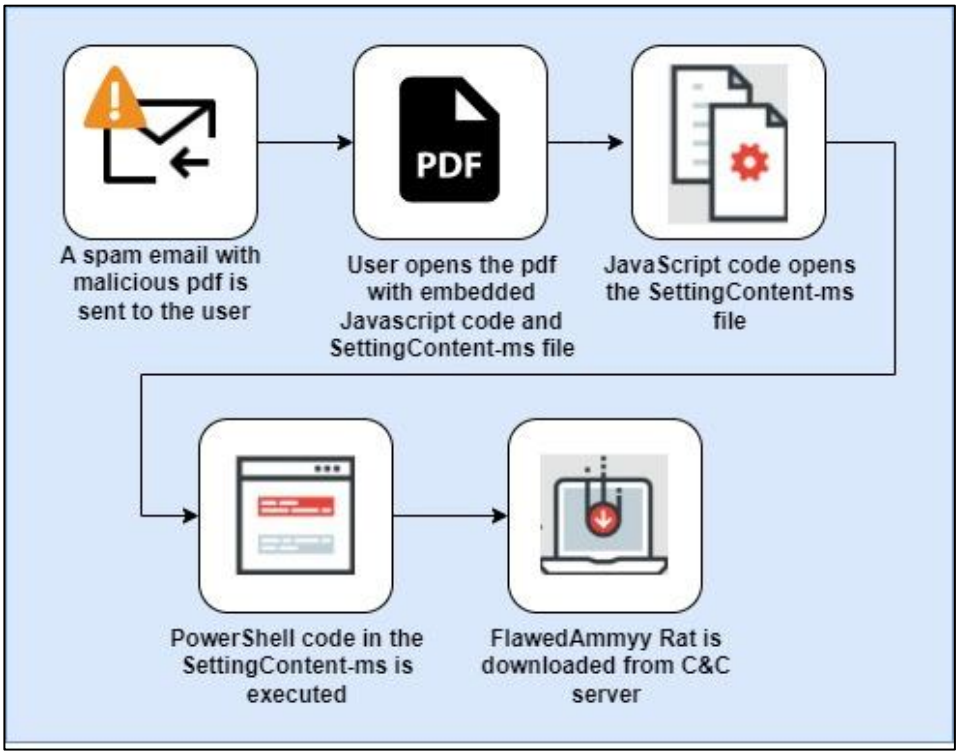


Figure 1: Attack vector

1.1 This presented solution tries to answer the Question:

How Can FIM system be optimized to detect the malicious contents in the files?

1.2 The Objective of the presented solution is to:

To develop and implement a solution for optimizing monitoring the integrity of files, which is lightweight and is not installed on kernel levels of the system. And to detect any files in the system and directories with their malicious content.

FIM plays an important role in compliance with CIS critical security controls which offer a structure for controlling cybersecurity risks and preventing attacks in environments that are on-premises, cloud-based, or hybrid. FIM specifically assists with the implementation of CIS

Control 4[3] . Currently, there are two ways to keep an eye on file integrity: real-time monitoring and periodic monitoring(Tang et al., 2014). These depend on the operating system and some like tripwire (Kim & Spafford) install as an agent or kernel module. Installing them on kernel level in an organization can expose the organization to potential risks. There are few solutions as the isolation aspect of the virtual machine (VM) architecture can strengthen the monitoring system, VM based security mechanisms offer a novel technique to lower these risks (Mishra et al., 2017; Win et al., 2014). Monitoring services are always included in the VM, a layer that sits between the upper operating system and the underlying hardware, in VM based systems. Thus, malware cannot identify or compromise them. Two categories of VM based file integrity monitoring solutions now in use are (Gupta S, 2012; Xiang et al., 2010) that demonstrate one approach, which checks the properties of the files and modification time. Another approach by (Asrigo et al., 2006). These file integrity monitoring tools are sophisticated and have semantic gap problems (Shi et al., 2018a) files, but cannot define what kind of file has been added to the system. Knowing what the file is and how it can harm the system is important for users to make better decisions in isolating and further analyse the attacked system. added to the system. Knowing what the file is and how it can harm the system is important for users to make better decisions in isolating and further analyse the attacked system.

This paper presents a programme for monitoring integrity of these system files and is also integrated to detect and report the type of files when any alerts is raised from the system regarding the modification of them. This solution helps in understanding any files in the system which can be malicious when executed, by scanning it before any actions are taken by the user. This method keeps the environment secure once any malicious activity is detected. Helping users to make better decisions with security factors. To achieve these results, the programme uses python as the base language, and for the integration part Yara rules are be used. The Yara rules can be created by the users to detect particular type of files which makes the system more secure and flexible. The programme is having some predefined Yara rules for demo purpose, but there is scope to add new ones for more functionality.

The remaining content is organised as follows: Section 2 gives a review of the previous work done in this field of study and a summarised table of the literature review. Section 3 gives a brief of the methodologies followed to derive the presented solution and gaps in previous work. Section 4 and 5 give the design specifications and implementation phase of the application with the challenges faced. Section 6 is about evaluating the presented solution and the experiments carried to optimize it to detect more malicious contents. In the final section 7 provides a summery of all the work completed and suggests potential future actions for future development of the work to identify a more ideal and optimised solution.

2 Related Work

This section presents an analysis of the research relating to subjects like various security testing methodologies being integrated into the FIM systems. The most popular and extensively utilised FIM systems are thoroughly examined to see if the added files scanning components

[3] https://www.cisecurity.org/controls/secure-configuration-of-enterprise-assets-and-software

can be integrated without violating the fundamentals of continuous verification and integrity checks(Kedgley, 2014; Wilbert & Chen, 2014).

2.1 Kernel level file integrity monitoring

There are a few systems like tripwire (G. H. Kim & Spafford, 1994), AIDE[4] (Advanced Intrusion Detection Environment), Osiris and Samhain (Wotring & Potter, 2005) which uses digital signature comparison to find potential changes to the files under observation. The downside is that they are all experiencing "delay detection" issues. Between the examination intervals, an intrusion could be launched by outsiders. There is not a perfect answer because this is a result of the designed scheme. In order to identify when a particular file is modified in real time, SNARE (System Intrusion Analysis and Reporting Environment), and I3FS (Patil et al., 2004) are file system layer implementations that intercept and trace down the associated VFS system calls. However, they require patching the monitored systems kernel, which is not always acceptable in the production system.

Higher assurance execution environments have been attempted to be built by a number of earlier systems in an effort to shield applications from malicious operating systems. To a certain extent, these techniques can also protect sensitive files. To ensure the integrity of the OS Kernel (Rhee et al., 2009) and (Xu et al., 2007) employ virtual machine introspection and interposition technologies to restrict access to sensitive kernel items within a single virtual machine. But while they shield kernel items like the interrupt table and kernel text, they are unable to stop unauthorised memory access to the file system cache and have the performance loss is quite evident. A few other initiatives, including (Azab et al., 2014; Santos et al., 2014) and (CriswellJohn et al., 2014) investigated transforming the original design into a protective mode. But these typically require significant adjustments to the way apps are developed and utilised. Such drastic changes present a significant obstacle to adoption (Shi et al., 2018b).

2.2 Virtual Machine File Integrity Monitoring

There are two prominent approaches for file system integrity monitoring in virtual machine environments, multi byte no operation (NOP) Injection technique and the Xen OS VMGuard file integrity monitoring solution. Them NOP injection addresses the challenges associated with virtual deployments. It is a debugger that makes use of breakpoints that are temporarily kept and watched by the gee compiler for NOP instructions. The lguest process thread administers NOP File System Integrity Tool (NOPFIT) processes, imposes security policies, and collects information about breakpoints. A kernel object parser keeps track of changes to the stack pointer inside the kernel. Similar approach by INT3FIT but uses 900 ms more than NOPFIT (J. Kim et al., 2010). But reliance on a debugger like NOPFIT introduces a potential single point of failure and may be susceptible to attacks targeting the debugging tool. Also, the injection technique adds complexity to the system, potentially increasing overhead, and resource utilization. And The approach may not be universally applicable, as it is tailored to specific virtual machine software and may not seamlessly integrate with other platforms.

[4] https://aide.github.io/

Another monitoring tool for virtual machines is XenFIT but needs to use Xen OS to use it. Their solution is native to the Xen environment and operates in real time. Breakpoints are used by XenFIT to identify when action has happened. After that, correlation is used to examine system calls made by the environment. Additionally, rather than storing the system attributes remotely, this approach uses a unique security policy for each device. The DomU kernel component can then be used by XenFit to gather data about the activity if a violation is discovered. Similarly VMGuard is a solution designed for virtual machines, The tool monitors privileged access to the virtual machine's management console (Domain0 in Xen) to detect and prevent malicious activities. VMGuard's architecture involves distributing predefined policies using GuardDomainU in trusted mode, followed by logging integrity measurements and comparing them to the latest environment measurements(Jin et al., 2010; Wang et al., 2012). The major limitation is that the solution is tightly coupled with the Xen operating system, limiting its applicability to other virtualization platforms. Also, the performance evaluation focuses on specific scenarios, and generalizability to diverse virtual machine use cases may require further exploration (Velten et al., 2013) (Fang et al., 2010).

2.3 Blockchain and Smart contracts

Blockchain makes it possible to create a decentralized database where organizations or institutions can conduct verified transactions without any party being able to exert control over the market. There are previous studies that have presented a solution that can check the integrity of files kept in the cloud by an external party without disclosing the contents of these files. The architecture in question offers a protocol built on challenges that result in a consistent and minimal usage of network bandwidth. Additionally, it provides a technique to balance the load of checks that accelerate or decelerate based on the behaviour of the storage service, all based on principles of computational trust (Pinheiro et al., 2018). However, the major limitations of these are the need for total faith in the third party service handling the integrity check of the files hosted in the cloud, there might be issues while conducting audits in terms of processes and outcomes. Also, it creates a major dependability on the storage service providers to maintain a service available 24 hours a day exclusively to receive the challenges submitted by the integrity check services (Pinheiro et al., 2020)(Pinheiro et al., 2021).

Table 1: Literature review

Title	Authers	Approach	Limitations
Kernel level file integrity monitoring	G. H. Kim & Spafford (1994), Wotring & Potter (2005), Patil et al. (2004), Rhee et al. (2009), Xu et al. (2007), Azab et al. (2014),	**Digital signature comparison**: Systems like Tripwire, AIDE, Osiris, and Samhain use digital signatures to compare and identify potential changes to monitored files. **File system layer implementations**: SNARE and I3FS intercept and trace VFS system calls to identify real time modifications.	Delayed detection issues in some of the systems. And kernel level installations for monitoring solutions. Also, performance loss observed in certain techniques and significant adjustments to application

	Santos et al. (2014), Criswell John et al. (2014), Shi et al.	**Higher assurance execution environments:** Techniques use virtual machine introspection and interposition to protect sensitive kernel items.	development and utilization
Virtual Machine File Integrity Monitoring	J. Kim et al. (2010), Fang et al. (2010)	**NOP Injection technique**: NOPFIT and INT3FIT use breakpoints to identify changes in virtual deployments. However, reliance on a debugger introduces potential single points of failure and may be susceptible to attacks. Injection technique adds complexity and may not be universally applicable. **Xen OS VMGuard:** XenFIT and VMGuard monitor file integrity in Xen environments	These techniques rely on a debugger that introduces potential single points of failure and may be susceptible to attacks. Injection technique adds complexity and may not be universally applicable. VMGuard is tightly coupled with Xen OS, limiting applicability to other virtualization platforms.
Blockchain and Smart contracts	Pinheiro et al. (2018, 2020)	**Blockchain-based solution**: Utilizes blockchain to create a decentralized database for file integrity checks in the cloud. The protocol addresses challenges in network bandwidth and load balancing based on principles of computational trust.	Dependencies on third party services for integrity checks. Issues in audit processes and outcomes and reliance on storage service providers for continuous availability

Research Niche: The presented technique is inspired from all the above works but is different in some aspects, the major one is that it is a very lightweight technique that continuously monitors the integrity of files, with minimum delay in notifying any modifications in the files and directories. Moreover, it uses a technique which detects if the files are malicious or not before executing or extracting them. There are few rules predefined while checking the file for any malicious contents but gives users an option to add more rules to precisely identify the files, avoiding any kinds of attacks resulting from a file addition.

3 Research Methodology

Based on the literature review there is no FIM technology that checks what files are added or detects any malicious contents from these added files. The research methodology for creating a solution that can check the contents in the files without opening it was solved using Yara rules. This solution involved several main phases, including an analysis of prior research in the field, a search for software-based solutions to the research aim, selection of the core issue type

that would be the focus of the remaining research cycle, a breakdown of file integrity monitoring systems requirements, the actual implementation of the application for detecting the changes, an assessment of the developed product, and additional aspects pertaining to future work and possible solutions.

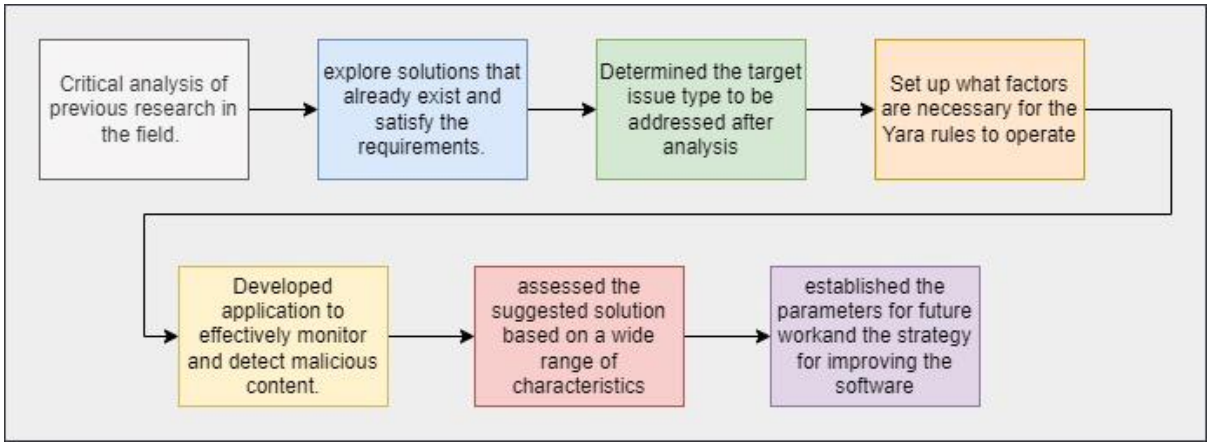


Figure 2: Research Methodology.

While Stages 5, 6, and 7 are covered in full in later sections of the research, the descriptive details for Stages 1-4 are provided in the section that follows. Stage 2 of the research process started with an analysis and determination of the current solutions in the market, whereas Stage 1 was covered in the earlier sections of the work (as part of the related work). As stated in the most recent part of the literature review, there is no solution which detects the contents in files added in the system.

3.1 Determining the gaps to be addressed

Whenever a file is added it is important for a user to get alert about it, as the file can be a malware with suspicious indent. There are many cases where malwares are executed just by opening a certain type of file. OWASP[5] has a separate blog post for unrestricted file uploads and the significant risks. So, this is important issue which can be integrated in file monitoring systems as it is effectively used in organizations and also comes under compliance and governance. Since to read a file and understand the malicious and non malicious contents in it, a solution was required which will solve it. To overcome this challenge the presented solution uses Yara rules with file integrity monitoring to detect these files additions and its contents. To configure this Yara must be installed on to the system. There is no application for performing any checks as Yara uses a CLI to perform the task. To check whether a file is malicious or not a particular type of malware was studied. This malware was a password stealer malware also known as fareit, the malware was analysed and the Yara rules were created according to the string available in the malware file. This made sure that when a Yara rule scans this malware it finds these strings or words in the malware and notifies the users with it.

[5] https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload a

3.2 Developing the application

As this file integrity monitoring continuously checks for modifications and changes in the file system and checks for newly added files, it must be lightweight, this is why a simple hash of files is saved for future comparison and a different loop for different functionalities were made. To create a file integrity monitoring system it needs to read the files from the system to start monitoring. To read files shell scripting is the best solution as shell scripts are handy for file and directory operations. They can be used to search, copy, move, delete, and manipulate files in a systematic way. This makes it beneficial for organizing and maintaining file systems. As this process runs in the background, Shell scripts allow users to automate repetitive tasks, reducing manual intervention and saving time. This is particularly valuable for tasks where the files are monitored continuously. But to start off with the implementation part, powershell was used as it gave a direct command line interface, a simple monitoring system that uses the secure hash algorithm (SHA256) algorithm to save hashes was designed using powershell which performed the basic tasks of creating a baseline and continuously monitoring the files, a baseline has all the hashes of the files saved before initializing the monitoring, and when the monitoring starts it uses this as the base for comparing the continuously extracted hashes, the solution was also successful in notifying the users about modifications and file addition or deletion. But when a file is added into the system it should be notified with a file added notification and the hash value of the file. In addition to this according to the presented solution it should also use Yara rules on the newly added files and define if they are malicious or not. When the implementation of this was initiated, there was a need for different methods to call which made it very complex to code it in powershell. Taking into consideration powershell was not a preferred language used as it had a limited number of libraries and methods to impose.

As there was a need for scripting languages with various features python language was used, the coding in python was started with the new purpose of solving the novelty problem of finding the hash and imposing Yara rules functions. Hashlib was used to create SHA256 hashes of the files as it's part of the python standard library, making it readily available without additional installations. Users can keep the same baseline if there are no changes in those files for a long period of time. However, if a user wants to change the file, they must create a new baseline before starting monitoring. This is to ensure that all the files have a new hash saved in baseline and are not compared with older ones giving false positive results when monitoring is started.

4 Design Specification

This section explains how a piece of built software can evaluate live scan results for a web product automatically, analyse them for later use, make necessary adjustments to address misconfiguration problems, and then retest the configurational file modifications. Section 4.1 provides more details about the roadmap workflow used in the software component's development. Section 4.2 describes the architecture of the FIM model and offers the option to include the software component from Section 4.1 in its design scheme.

4.1 Yara Rules creation

A process roadmap was made, as seen in Figure 5, in order to specify the precise aspects of the implementation and to reduce the scope of the research. Choosing a method that has scanning features of the file and presenting them, deciding on a testing object or malware, selecting a tool to analyse the malware for testing it with Yara rules and then implementing a software solution taking considerations of all the metrics that could be used to show the advantages of the developed approach were the main steps of the research route.

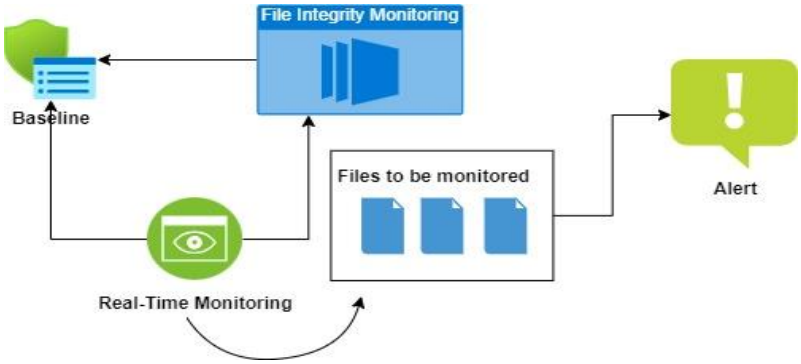


Figure 3: Roadmap to specify the workflow of FIM.

Although most of the above phases have already been covered in the earlier portions of this work, the focus was placed on analysing the capacities of Yara rules which provide a brief of different malwares according to the created rules. With regards to this the file hash of this malware is also presented so that it can be checked for any previous histories. As a result the final architecture is described in figure 4 in which whenever a file is inserted Yara rules gets activated and scans it immediately for its contents. The remaining tasks rely on how the Yara rules are configured to detect the malware contents in the file.

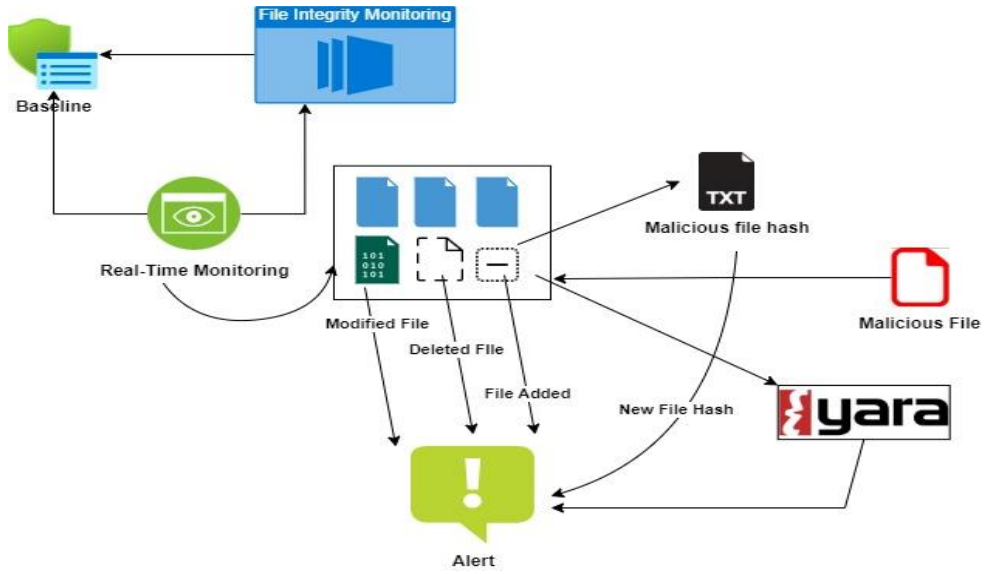


Figure 4: Architecture of the FIM system developed.

5 Implementation

File integrity monitoring is used on a wider scale from small businesses to top tier companies, this ensures the integrity of the file systems. There are tools available in the market that are used by the organization, but they lag in detecting or notifying the users. Moreover, this can put the organization at potential risk with crucial files in the system. to overcome these issues, an in house FIM solution is needed which can be lightweight and efficient. To monitor the system files power shell was used, but the low availability of various features and libraries made it difficult to assign all the features to this system. Python which is another shell language was used in a later stage where all the features were tested before compiling them in a single code.

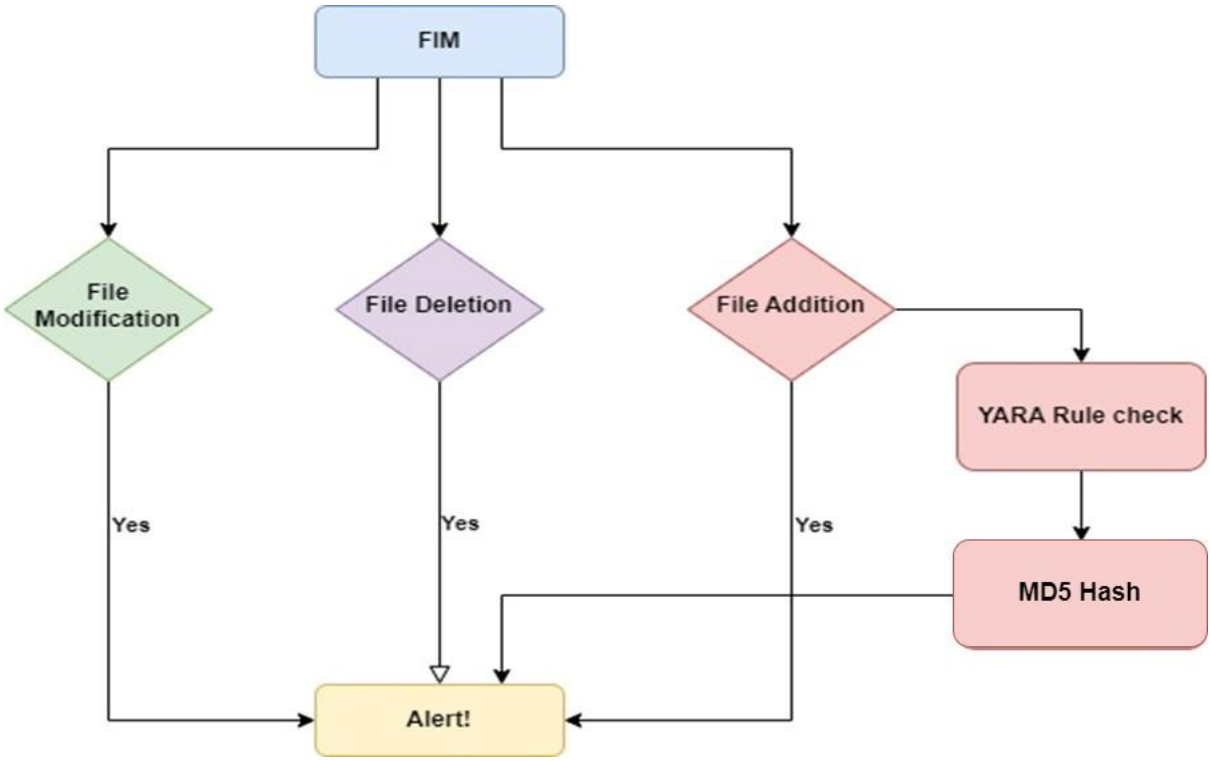


Figure 4: Flow Diagram

Before starting the monitoring part, the main thing is to create a baseline of files we want to monitor. For instance, if there is a need to monitor files in a particular directory, it is fed to the application. Once done a baseline needs to be created which saves the hash values of the desired files in it, which is used while monitoring to compare it with the latest hash values. A simple change in the files drastically change the hash values of these files. The baseline has hash values and the path of the file, which will help while comparing of file hashes, and creates no problems with understanding what hash belongs to what file. When monitoring is started and there is a change in any of these files the compare function won't recognize the hash values and notify the user about the changes in the system. The comparing function is set to compare the hash values for every second by default but can be changed according to the importance of the files. If there is any deletion of files from the selected directory, the compare function will read a null value for the scanned files with the baseline and hence notify the user about the deletion of the file. If there is an addition of files in the system, users not only get notified about it but also get the hash value of the added value. Users can then check the hash values on online

websites to understand if the added file is malicious or not. If the file is malicious further actions like isolating the machine can be initiated by the users. But what if the malware is a new version and the hash is not yet listed on the websites, to overcome this problem the application has a set of solutions that it implements as soon as the file has been added to the system. The application has some predefined rules by which it checks what is there in the file added. These rules can be modified according to the organization's architecture and the file systems. These rules also known as Yara rules can be very beneficial as they are able to scan a file without opening or executing it. It uses a search system where it checks for malicious content in the files and gives an output accordingly. It notifies what are all the malicious contents in the file and gives a score so the users can act as per the protocols.

6 Evaluation

As all the system is up and running any changes in the file system are reported to the users. While creating the baseline it accurately captures the hash values and file paths of the desired files, saving them in the text file for future use. Analysis was performed to check the precision and accuracy of the presented monitoring system. High precision and recall values indicate a robust monitoring system with minimal false positives and false negatives. This underscores the reliability of the system in capturing file changes accurately, establishing a solid foundation for practitioners relying on precise file integrity monitoring.

The notification module's real time alerting capabilities were thoroughly examined under varying test scenarios to assess the timing and accuracy of notifications. Response time metrics, including notification delivery time and the time taken to identify file changes, underwent statistical analysis. Average response times were compared against defined benchmarks to ascertain the system's responsiveness, as soon as there was any change the notification timings were around 30 milliseconds, which is much better than other systems. The system exhibited prompt notification delivery, meeting, or surpassing industry standards for real time responsiveness. This finding is important for users who rely on timely alerts to address security incidents instantly and efficiently.

The Yara rule engine, designed for proactive threat detection, underwent evaluation by introducing files with known malicious content and assessing the systems ability to identify and score threats accurately. Effectiveness metrics, including true positive, true negative, false positive, and false negative rates, were employed for better understanding of the Yara rule capability to distinguish between malicious and non malicious content. When experimenting with password stealer malwares the true positives were 80 and false positives were 10, the false negatives were around 5, these overall results made the precision as 0.88 and recall value as 0.94 making the f1 score as 0.91.

This value indicates a balance between precision and recall, and a higher F1 score suggests a better performing model. This capability enhances the systems proactive defence, providing users with a powerful tool for threat detection. Few experiments were performed to get the accurate observations from the system. It was observed that more precise the Yara rules are created, the system was significantly able to detect the malicious files. The significant experiments are mentioned below.

6.1 Experiment 1

yararule.yara

```
rule creds_ru
{
meta:
    description = "Simple YARA rule"

strings:
    $a= "msi.dll"
    $b= "POP3 passwords"
    $c= "SelmpersonatePrivilege"
    $d= "SeChange NotifyPrivilege"
    $e= "SeCreateTokenPrivilege"



condition:
    ($a or $b or $c or $d or $e)
}

import "math"
import "pe"

rule MAL_Backdoor_DLL_Nov23_1_RID2F31 : CVE_2023_4966 DEMO EXE EXPLOIT FILE MAL RANSOM {
   meta:
      description = "Detects a backdoor DLL, that was seen being used by LockBit 3.0 affiliates exploiting CVE-2023-4966"
      author = "X__Junior"
      reference = "https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-325a"
      date = "2023-11-23 11:46:41"
      score = 80
      customer = "demo"
      license = "CC-BY-NC https://creativecommons.org/licenses/by-nc/4.0/"
      hash1 = "cc21c77e1ee7e916c9c48194fad083b2d4b2023df703e544ffb2d6a0bfc90a63"
      hash2 = "0eb66eebb9b4d671f759fb2e8b239e8a6ab193a732da8583e6e8721a2670a96d"
      tags = "CVE_2023_4966, DEMO, EXE, EXPLOIT, FILE, MAL, RANSOM"
      minimum_yara = "1.7"
```

Figure 5: Yara Rule 1

In this experiment, specific types of malwares were used like password stealers or spywares. As there were few Yara rules predefined in this research, in the testing phase whenever a file was added it scanned the file with this predefined Yara rule. Whenever a file is added the SHA256 hash is saved in a document, this was very precise as whenever a file was added, even the non malicious file hash was saved there, which was used for checking it on websites like Virus total. To create a Yara rule few of the malwares were studied and the rules were created accordingly. Whenever a malware interacts with these predefined rules is added purposely while experimenting, the Yara rules did their work giving the notification with results of detected strings or contents. For instance, when a password stealer malware was added to the monitoring directory, there was a notification stating new file added, but the Yara rules detected the contents from the file with malicious strings and notified them as well. Similar such files were added to the monitoring directories and according to the rules added the files were shown their malicious content for which they were filtered out. For every similar password stealer malware added, the Yara rules were able to detect them. Whenever a malware was added and the Yara rules were not able to detect them, the hash was checked, as for every added file hash is calculated and saved in a different file. This de it easier to find the known malwares. But when new malwares or distinct types of malwares were added, hash values justified them, but Yara rules did not detect them, so a new experiment was initiated.

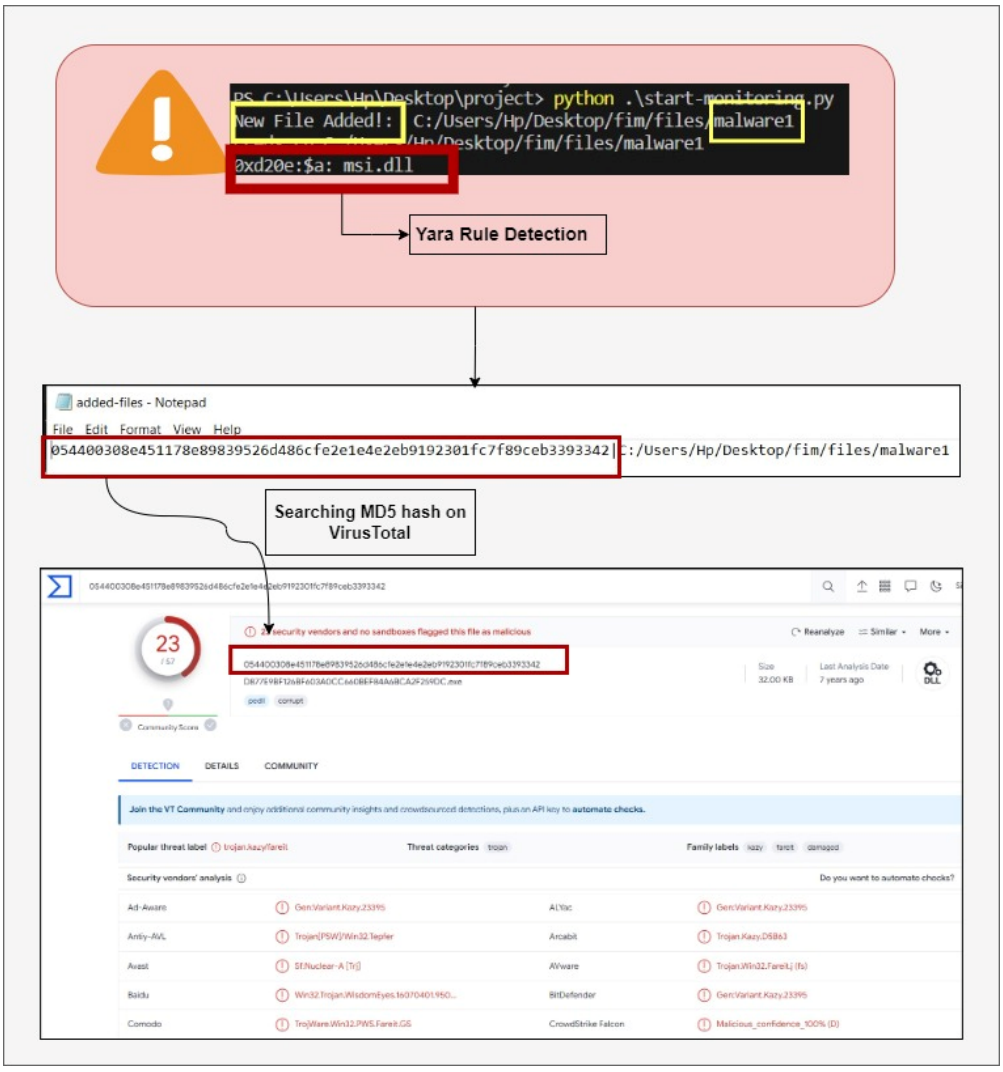


Figure 6: Yara rule detecting added malware.

6.2 Experiment 2

This experiment was performed to check if the Yara rules when redefined are able to detect more numbers of malwares. When diverse types of malwares were checked with the monitoring system, only the rules which were catching them were reported as malicious, but few malwares and shells went undetected from the Yara rules. Although the hashes were compared to check if those are malicious or not, as the technology is evolving many malwares go undetected. There might be cases in an organization where new malware is used to attack and the hash values are not available on these websites like Virus total. Or there can be a situation of shell attacks where a file hash would not be sufficient. In this experiment, many new malwares were taken, and these went undetected, so the Yara rules were added with the previous rules according to the newer added malwares. When these experiments were again performed, the Yara rules were able to correctly detect them.

```
    $e= "SeCreateTokenPrivilege"
    $f= "Delete Object"
    $g= "GetOEMCP"
    $h= "GetCPinfo"
    $i= "ReadConsole"
    $j= "WriteConsole"
    $k= "CreateProcess"
    $l= "GetCommandLine"


condition:
    ($a or $b or $c or $d or $e or $f or $g or $h or $i or $j or $k or $l)
}
```

Figure 7: Adding more Yara rules to the file.

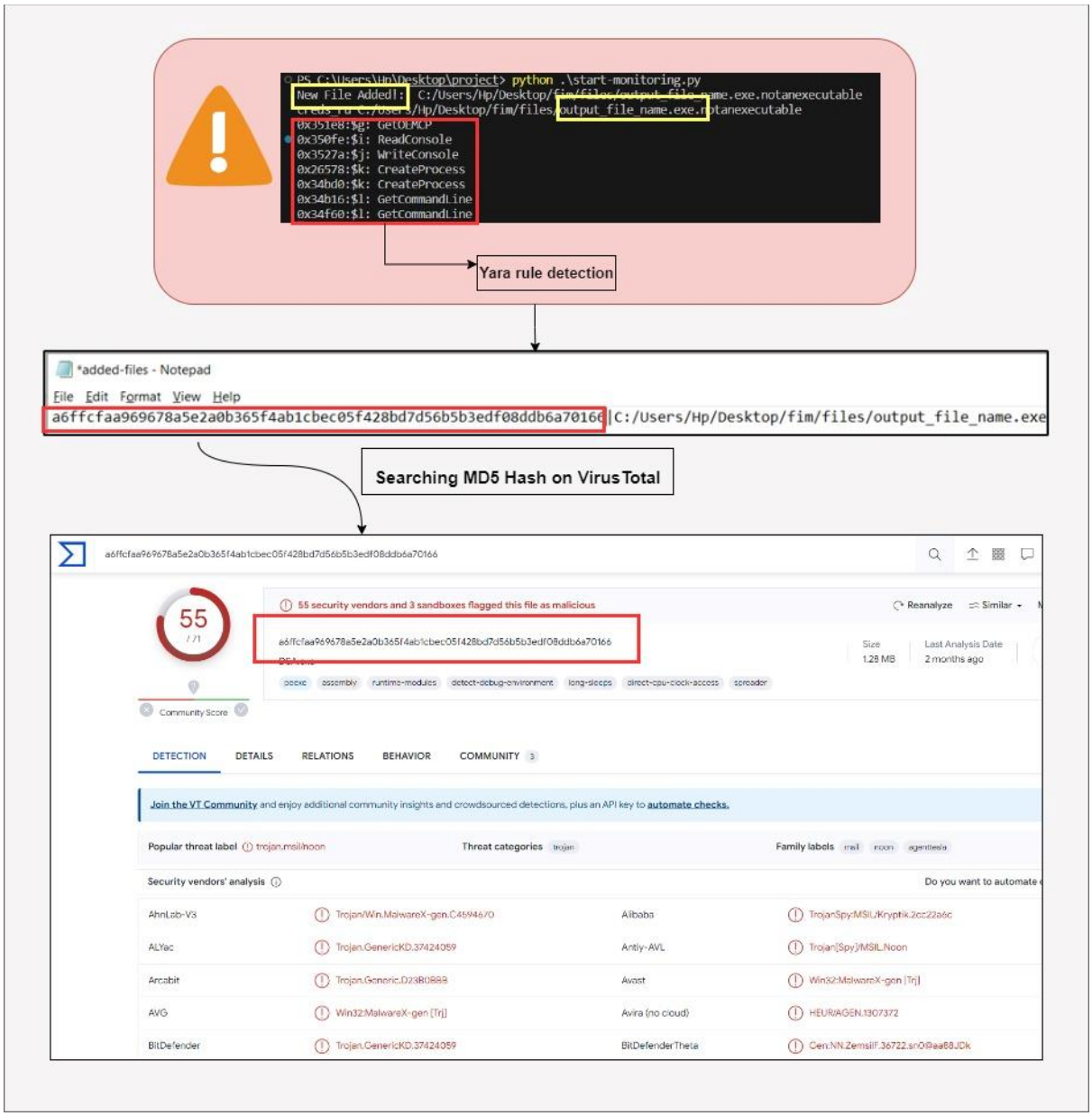


Figure 8: Redefined Yara rule detecting newly added malware.

6.3 Discussion

While experimenting basic functionalities, the file integrity monitoring system precisely notifies the modifications and deletions part of the part with minimum delays. The simple modifications like adding s string in a file, has also been detected and the hash is drastically changed, and the compare functions precisely detects it triggering the notification. As soon as the null value is found by this compare function it understands that a file has been deleted and notifies it as well. When new files are added the systems gets proactive taking the hash of that file and notifying the user about file addition and the corresponding file hash. The users can check this hash value on websites like Virus total and the Yara rules to find the contents of the file so users can recognize if the file is malicious or not.

The crux of the challenge is encapsulated in formulating Yara rules; the greater the specificity of these rules, the more accurate the detection outcomes will be. There can be cases where false positives are notified by these Yara rules. This situation may arise when these rules are over defined, like in a situation where the user adds a genuine file to the system and the Yara rules can find some strings or contents in that file triggering a false alert. So, to overcome these types of changes the developers can use secure coding languages and can check for the defined Yara rules to avoid the false alerts. The users can also stop the monitoring system and add complete the desired changes. Once all the changes are completed new baseline should be created before monitoring or else the system will use the older baseline to check these newer files and give false alerts. So according to the literature review this system not only acts like a file integrity monitoring system but also tells weather the added files are malicious or not. The only limitation is with the optimization of Yara rules, more specific these rules are, the system will be more accurate.

7 Conclusion and Future Work

The objective of this research is fulfilled as the system works with the f1 score of 0.91. The solution was precise enough to work as a normal file integrity monitoring solution but was also able to detect and alert the users with malicious file additions. The f1 score or the accuracy of the system can be increased by creating the Yara rules which are specifically designed according to the need of organization, for example if an organization feels that they can be a victim of password stealer malwares, the Yara rules should be optimized according to those malware files. The key finding of this solution is that Yara rules are pretty good in checking the contents of any files without ever opening or executing them, and integrating these rules with file integrity monitoring gives us a secure solution to defend against malware attacks. The system was efficient enough as it was able to notify the user within a span of 30 milliseconds and gave the correct outputs for all the basic functionalities of the system. The only limitation lies in defining the Yara rules by the users, more specific the Yara rules, more malicious content is notified. The future work on this system can be implemented on securing baseline, for databases which are just used to store the backup files, the baseline of these can be pushed to blockchain securing it and there will be no possibility for an attacker to change the baseline. The current system does not focus on securing a baseline, but this method can be implemented to secure it. The system is not capable of securing files which are updated periodically like log

files. As these files are updated again and again the hash of these file keeps changing, so a method that can save the temporary image of the file and compare it with previous one can be implemented. Newer integration of features can also be implemented to make it more secure and precise in detecting malicious files.

References

Asrigo, K., Litty, L., & Lie, D. (2006). Using VMM-based sensors to monitor honeypots. *VEE 2006 - Proceedings of the Second International Conference on Virtual Execution Environments*, *2006*, 13–23. https://doi.org/10.1145/1134760.1134765

Azab, A. M., Ning, P., Shah, J., Chen, Q., Bhutkar, R., Ganesh, G., Ma, J., & Shen, W. (2014). Hypervision across worlds: Real-time kernel protection from the ARM trustzone secure world. *Proceedings of the ACM Conference on Computer and Communications Security*, 90–102. https://doi.org/10.1145/2660267.2660350

CriswellJohn, DautenhahnNathan, & AdveVikram. (2014). Virtual ghost. *ACM SIGARCH Computer Architecture News*, *42*(1), 81–96. https://doi.org/10.1145/2654822.2541986

Fang, H., Zhao, Y., Zang, H., Huang, H. H., Song, Y., Sun, Y., & Liu, Z. (2010). VMGuard: An integrity monitoring system for management virtual machines. *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 67–74. https://doi.org/10.1109/ICPADS.2010.44

Gupta S. (2012, December). *A light weight centralized file monitoring approach for securing files in Cloud environment | Request PDF*. Internet Technology And Secured Transactions, 2012 International Conference For. https://www.researchgate.net/publication/261051009_A_light_weight_centralized_file_monitoring_approach_for_securing_files_in_Cloud_environment

Jin, H., Xiang, G., Zou, D., Zhao, F., Li, M., & Yu, C. (2010). A guest-transparent file integrity monitoring method in virtualization environment. *Computers and Mathematics with Applications*, *60*(2), 256–266. https://doi.org/10.1016/J.CAMWA.2010.01.007

Kedgley, M. (2014). File integrity monitoring in the modern threat landscape. *Network Security*, *2014*(2), 5–8. https://doi.org/10.1016/S1353-4858(14)70019-4

Kim, G. H., & Spafford, E. H. (1994). The design and implementation of Tripwire: A file system integrity checker. *Proceedings of the ACM Conference on Computer and Communications Security*, 18–29. https://doi.org/10.1145/191177.191183

Kim, J., Kim, I., & Eom, Y. I. (2010). NOPFIT: File system integrity tool for virtual machine using multi-byte NOP injection. *Proceedings - 2010 10th International Conference on Computational Science and Its Applications, ICCSA 2010*, 335–338. https://doi.org/10.1109/ICCSA.2010.79

Mishra, P., Pilli, E. S., Varadharajan, V., & Tupakula, U. (2017). Intrusion detection techniques in cloud environment: A survey. *Journal of Network and Computer Applications*, *77*, 18–47. https://doi.org/10.1016/J.JNCA.2016.10.015

Patil, S., Kashyap, A., Sivathanu, G., & Zadok, E. (2004). I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System. *LiSA*.

Pinheiro, A., Canedo, E. D., Albuquerque, R. de O., & de Sousa Júnior, R. T. (2021). Validation of architecture effectiveness for the continuous monitoring of file integrity stored in the cloud using blockchain and smart contracts. *Sensors*, *21*(13). https://doi.org/10.3390/S21134440

Pinheiro, A., Canedo, E. D., De Sousa Junior, R. T., Albuquerque, R. D. O., Villalba, L. J. G., & Kim, T. H. (2018). Security Architecture and Protocol for Trust Verifications Regarding the Integrity of Files Stored in Cloud Services. *Sensors (Basel, Switzerland)*, *18*(3), 4–7. https://doi.org/10.3390/S18030753

Pinheiro, A., Canedo, E. D., Sousa, R. T. De, & Albuquerque, R. D. O. (2020). Monitoring file integrity using blockchain and smart contracts. *IEEE Access*, *8*, 198548–198579. https://doi.org/10.1109/ACCESS.2020.3035271

Rhee, J., Riley, R., Xu, D., & Jiang, X. (2009). Defeating dynamic data kernel rootkit attacks via VMM-based guest-transparent monitoring. *Proceedings - International Conference on Availability, Reliability and Security, ARES 2009*, 74–81. https://doi.org/10.1109/ARES.2009.116

Santos, N., Raj, H., Saroiu, S., & Wolman, A. (2014). Using ARM TrustZone to build a Trusted Language Runtime for mobile applications. *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 67–80. https://doi.org/10.1145/2541940.2541949

Shi, B., Li, B., Cui, L., & Ouyang, L. (2018a). Vanguard: A cache-level sensitive file integrity monitoring system in virtual machine environment. *IEEE Access*, *6*, 38567–38577. https://doi.org/10.1109/ACCESS.2018.2851192

Shi, B., Li, B., Cui, L., & Ouyang, L. (2018b). Vanguard: A cache-level sensitive file integrity monitoring system in virtual machine environment. *IEEE Access*, *6*, 38567–38577. https://doi.org/10.1109/ACCESS.2018.2851192

Tang, X. J., Lu, Y., & Liu, N. (2014). Design and Implementation for File Monitor System Based on Windows Driver. *Proceedings - International Symposium on Parallel Architectures, Algorithms and Programming, PAAP*, 289–292. https://doi.org/10.1109/PAAP.2014.31

Velten, M., Wessel, S., Stumpf, F., & Eckert, C. (2013). Active file integrity monitoring using paravirtualized filesystems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8292 LNCS*, 53–69. https://doi.org/10.1007/978-3-319-03491-1_4

Wang, Z., Huang, T., & Wen, S. (2012). A file integrity monitoring system based on virtual machine. *Proceedings of the 2012 2nd International Conference on Instrumentation and Measurement, Computer, Communication and Control, IMCCC 2012*, 653–655. https://doi.org/10.1109/IMCCC.2012.396

Wilbert, B., & Chen, L. (2014). Comparison of File Integrity Monitoring (FIM) techniques for small business networks. *5th International Conference on Computing Communication and Networking Technologies, ICCCNT 2014*. https://doi.org/10.1109/ICCCNT.2014.6963090

Win, T. Y., Tianfield, H., & Mair, Q. (2014). Virtualization security combining mandatory access control and virtual machine introspection. *Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014*, 1004–1009. https://doi.org/10.1109/UCC.2014.165

Wotring, B., & Potter, B. (2005). Host Integrity Monitoring Using Osiris and Samhain. *Host Integrity Monitoring Using Osiris and Samhain*, 1–421. https://doi.org/10.1016/B978-1-59749-018-4.X5000-X

Xiang, G., Jin, H., Zou, D., Zhang, X., Wen, S., & Zhao, F. (2010). VMDriver: A driver-based monitoring mechanism for virtualization. *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, 72–81. https://doi.org/10.1109/SRDS.2010.38

Xu, M., Jiang, X., Sandhu, R., & Zhang, X. (2007). Towards a VMM-based usage control framework for OS kernel integrity protection. *Proceedings of ACM Symposium on Access Control Models and Technologies, SACMAT*, 71–80. https://doi.org/10.1145/1266840.1266852