2022 2nd International Conference on Digital Futures and Transformative Technologies (ICoDT2) | 978-1-6654-9819-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICoDT255437.2022.9787428

File Integrity Checkers: Functionality, Attacks, and Protection

Ahmed Salman
Department of Information Security
Military College of Signals
Rawalpindi, Pakistan
ahmedsalman.82@gmail.com

Muhammad Sohaib Khan
Department of Information Security
Military College of Signals
Rawalpindi, Pakistan
sohaib.niazi@mcs.edu.pk

Sarmad Idrees
Department of Information Security
Military College of Signals
Rawalpindi, Pakistan
sidrees.msis-19mcs@student.nust.edu.pk

Faisal Akram
Department of Information Security
Military College of Signals
Rawalpindi, Pakistan
faisal.akram@mcs.edu.pk

Muhammad Junaid
Department of Information Security
Military College of Signals
Rawalpindi, Pakistan
muhammadjunaid@mcs.edu.pk

Aamer Latif Malik
Department of Information Security
Military College of Signals
Rawalpindi, Pakistan
aamerlatifmalik@gmail.com

***Abstract*—Intrusion detection systems are a critical component of a network's security. Intrusion detection systems exist in a variety of shapes and sizes, with various methodologies and analytic procedures. Host-based intrusion detection systems, or HIDS, are intrusion detection systems that operate at the host level and use a signature database (DB) or a profile to do detection analysis. The integrity of the database is fully dependent on the detection in all host-based systems. If an attacker can edit the database to his liking, he can simply circumvent the HIDS. In this paper, we have focused our study on file integrity checking HIDS. An endeavor has been made to study this specific type of HIDs functionality and various attacks against its trusted operation. Different techniques used to secure the system database have been studied, however, none are found to be flawless. We conclude that the use of Blockchain can be a viable solution in the future to secure the critical database integrity in such systems.**

***Index Terms*—HIDS, File integrity checkers, Blockchain**

I. Introduction

Generally, HIDS is not considered an active area of research. The reasons for less research and arguments to perform greater research on such systems have been amply covered by the authors in [1]. This paper focuses on the study of file integrity checking HIDS or File Integrity Monitors (FIMs). We know that a file system is a useful data source for intrusion analysis [2] and FIMs maintain a database of file signatures for their protection operation. The reasons to select file integrity checking HIDS for our research on DB protection are:

- Simpler design requirements as evident in Fig. 1, which can help focus research on the integrity protection issue..
- Since the DB must be built from the file system, no training dataset or time is required.
- No external data, such as virus signatures, is necessary to update the database.

Although file integrity checking is not a glamourous study topic, it is still crucial. Before getting into further depth on DB protection and other topics, we'll go over some additional material on the necessity of file integrity checks HIDS.

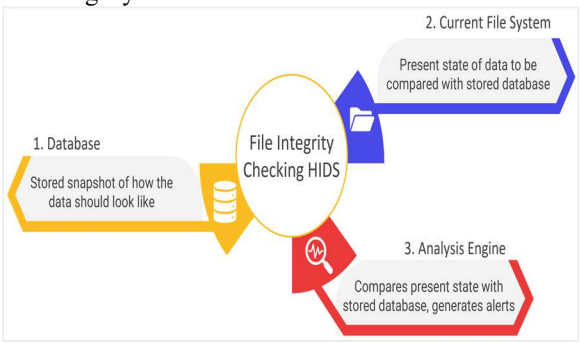

Fig. 1: Components of file integrity checking HIDS

A. Importance of File Integrity IDS

File integrity IDS has been considered an essential part of information security best practices. Almost all security standards promote the use of integrity checkers, but PCI DSS is the most popular of them all. PCI DSS recommends using file integrity monitoring in its clause no 11.5 to protect against unauthorized modifications of critical files. There is a constant debate that anti-virus can perform the same functions as file integrity checkers however, the argument is not that simple. Although an anti-virus is quite effective in removing malware on introduction to a system but only when a valid signature or a profile is available. A file integrity checker, on the other hand, does not require a signature database or a profile database. Rather, it can use its own integrity database to identify

any file system changes. As a result, HIDS is more effective in detecting zero-day malware.

The principle of defense in depth is also implemented by using a file integrity checker. Due to the current concealment tactics, a cautious intruder may never be caught if a computer without an integrity checker is hacked. In this instance, however, certain files would be modified, which a file integrity IDS can readily identify.

B. HIDS functionality and issues

Authors in [3], divide the HIDS capabilities into three groups. Perimeter control, host access control, and a change control mechanism are among them. Intruders may simply overcome all these functions, according to the authors, if specific conditions are met. The authors advocate employing file integrity checking HIDS to assure the integrity of the targeted system as a solution to these assaults. They also refer to such systems as *Change Auditing Systems*.

C. Fundamental qualities of file integrity checking HIDS

Authors in [4] have conducted a thorough investigation on the various types of file integrity monitors available on the market. They provide a list of fundamental criteria in a FIMs before examining the kinds, inspired by work in [5], which are: -

1) Checks on meta information: As a trusted non-super user executable may fall into the category of a non-trusted executable for the root or superuser, file integrity monitors must additionally examine the Metadata of a file in addition to the file contents.

2) Automated process: The system must not rely on user input to commence reaction action, otherwise the malicious activity will be granted large harm space.

3) Self-Protected: If an attacker modifies system files, it indicates that he has achieved elevated access on the machine. Modifying the database is therefore a simple job. To provide secure identification of malicious behavior, measures to maintain the HIDS DB must be in place.

4) Relevance: Automation generates a large amount of output data. Users can then utilize this information to assess attacks and make other crucial decisions. The greater the volume of output, the greater the possibility of human error. As a result, the output must only be useful and relevant to the human user.

5) Repetitive checking: The majority of FIMs are periodic HIDS, which allows the attacker to edit files and execute activities to disguise himself. Such methods are ineffective compared to real-time systems that can manage access requests based on security policies.

6) Upgradeable systems: To protect against new vulnerabilities, a system is updated with fresh updates. Even trusted executables will not be executed if the related file integrity checking HIDS is not upgraded along with the new file integrity DB. As a result, a file integrity checker must be upgradeable with minimal or no human involvement. The authors of [6] distinguish between 'weak' and 'strong' intrusions. A strong intrusion occurs when an invader is able to colonize a system, whereas a weak intrusion occurs when the intruder is unable to totally take over the system. A competent file integrity checker with the criteria listed above lowers a strong incursion to a weak intrusion over time.

II. TYPES OF FILE INTEGRITY CHECKING HIDS

We'll utilise Linux OS terminology to explain the functions and levels of operations when exploring different file integrity checks HIDS. Because Linux is an open standard that can be easily adjusted, a fruitful conversation about how to create a perfect file integrity monitor may be developed. As previously indicated, the authors of [4] conducted a thorough assessment of File Integrity HIDS kinds. The file integrity checking HIDS has been separated in terms of its operation location. The following types have been defined: -

A. User space checkers

User-space systems are substantially more vulnerable than kernel-space systems because they are more susceptible to malicious intruder alterations. These rely significantly on the system's safety measures for protection. As a result, such systems have legitimate security concerns. Furthermore, because these systems cannot intercept calls at lower OS levels, they must perform frequent integrity checks. Furthermore, this type of periodic monitoring takes a toll on the system's performance with each check. However, because of the libraries and resources accessible at this level, they are significantly easier to develop and function in the same way as an application. Tripwire [7] is the most popular FIM among users. YAFIC, AFICK, and AIDE are some more instances.

B. Nonresident checkers

These systems are housed in their own system and monitor the integrity of one or more other systems. In the sense that hosts have no way of intruding into the server or remote HIDS system, these systems are secure. However, there are a few

issues with it, such as what the host should do if the connection between the hosts and the server is broken, or a period between checks that provides opportunity gaps for attackers while still protecting the client software from penetration. Osiris is a prominent non-resident checker, but others include Radmind, Veracity, and Samhain.

C. Kernel space checkers

These systems are more complicated to create and develop. If not correctly constructed, they also have a considerable performance overhead. However, such systems are better than any other type for integrity verification. Because the system is at the lowest level of the operating system, it is not totally reliant on system defences. Because all file requests must pass via the kernel, it may enable real-time integrity monitoring. Furthermore, based on the security policy, it might provide or refuse access to files. I3FS, DigSig, WLF, and the SOFFIC umbrella are some examples.

D. Kernel and User space hybrid checkers

Such systems examine files in both kernel and user space. [8] describes one such system as an example. A hybrid checker can combine the benefits of both types of file integrity checks, such as examining files in various executable formats such as ELF and others. However, keeping an updated database to verify all formats is tough. Additionally, the security of communications between the kernel and user space might be jeopardized.

III. ATTACKS AGAINST FILE INTEGRITY CHECKING HIDS

In [9], the authors present extremely specific obstacles for file integrity checks and how to overcome them. Although there have been detailed challenges to HIDS security, we will solely focus on file integrity checking HIDS here. The file integrity HIDS has been divided into three modes by the authors: initialize mode, check mode, and update mode.. Attacks have been given against each as follows: -

A. Attacks against Initialize mode

In initialize mode, if a system is already hacked, a compromised database will be built and utilized in following phases of operation. To ensure the absence of any compromised system component, run the HIDs immediately after installing the OS and in offline mode.

B. Attacks against Update mode

IAn attacker can execute the update mode on a changed machine or just manipulate the database in base level integrity checking tools. However, if the database is kept on the same machine, this can happen. The problem might be solved by password-protecting the update mode, encrypting the database, or storing the database at a distant site. This can guard against both insider and physical threats.

C. Attack against Check mode

This mode's attacks come in a variety of forms. Some are listed here: -

1) Trojan Binary: An intruder can replace the standard system binary that checks integrity with a malicious one that reports proper file results. Own hash checking binary is used as a countermeasure.

2) Loadable Kernel Module: An intruder can use his own LKM to hijack system calls and send them to a malicious application that is stored elsewhere. More checks can be implemented to prevent the attacker from remapping a larger number of system calls, which is not straightforward.

3) Faked Report: An attacker can pass a bogus report over the reporting channel between the checking and reporting modules. He can also use an OK report generator to replace the system binary. This may be avoided by establishing encrypted channels between modules or by validating the integrity of the system binary.

4) Checking Gap attack: An adept attacker can complete his task swiftly by checking HIDS on a regular basis and replacing the malicious file with the original when it has been completed. Because it is illogical to run periodic checks at such short intervals, this is the most likely attack vector. To defend against this attack, use kernel level checkers to intercept file access requests and verify integrity on the fly. There are other daemon checkers, such as Samhain, that operate in the background.

D. Database Protection

As we've seen, the placement of HIDS, as well as the HIDS database and database integrity, are critical for the system's health and proper operation. The authors of [4] recommend that the integrity checker be placed at the kernel level and that it only check certain files for efficiency. It provides a few choices for database security, such as saving the database offline on external write-only storage or on the hard disc. External media is safer, but it is more difficult to update. The authors of [10] go into further detail on DB protection. They define the following three strategies for DB integrity protection: -

TABLE I: Comparison of Blockchain and central DB [11]

	Permission less Blockchains	Permissioned Blockchains	Centralized Database
Throughput	Low	High	Very High
Latency	Slow	Medium	Fast
Readers	High	High	High
Writers	High	Low	High
Untrusted writers	High	Low	0
Consensus protocol	PoW, some PoS	BFT protocol [12]	Nil
Central Management	No	Yes	Yes

A. *Read only Media*

Storing the database on a write-protected CD or USB can make it necessary for the user to have physical access to the database. It is useful in computers with few changes in database files; however, updating in a frequently changing environment requires a lot of time.

B. *Use of Signature*

Information about the file to maintain the integrity of the database, it can be safeguarded using a hash function or a digital signature. This is more efficient in terms of file updates, but the encryption overhead increases system complexity..

C. *Remote DB Storage*

The database can be hosted on a distant server and accessed securely. This has operational benefits since all hosts may be updated for policy and protected from a single server. However, if the server is hacked, it still creates a single point of failure.

IV. USING BLOCKCHAIN FOR DB PROTECTION

Following the explanation above, it is evident that none of the options to safeguard DB are perfect and contain some tradeoffs. The usage of Blockchain to ensure database integrity is one way to DB protection. As a test case, we address DB security in the context of file integrity checking HIDS; nevertheless, we believe that this technique may be applied to any HIDS or NIDS that maintains some form of DB for detection. To confirm our strategy, we'll apply the method described in [11] to figure out which Blockchain solution is best for our Host-based integrity checkers. The authors present their findings in Fig 2 as a flowchart. Most people think of Blockchain as a storage technology, however it is actually the world's worst database. The primary benefit of Blockchain is *Immutability* and *Decentralization*. As we can see from Fig 2, For HIDS database integrity security, Blockchain can be used. To avoid the complexity of an online Blockchain presence, we might utilize *Private Permissioned Blockchain* for the initial test case. Table I compares permissioned, permissionless, and central DB Blockchains to put technical features into context.

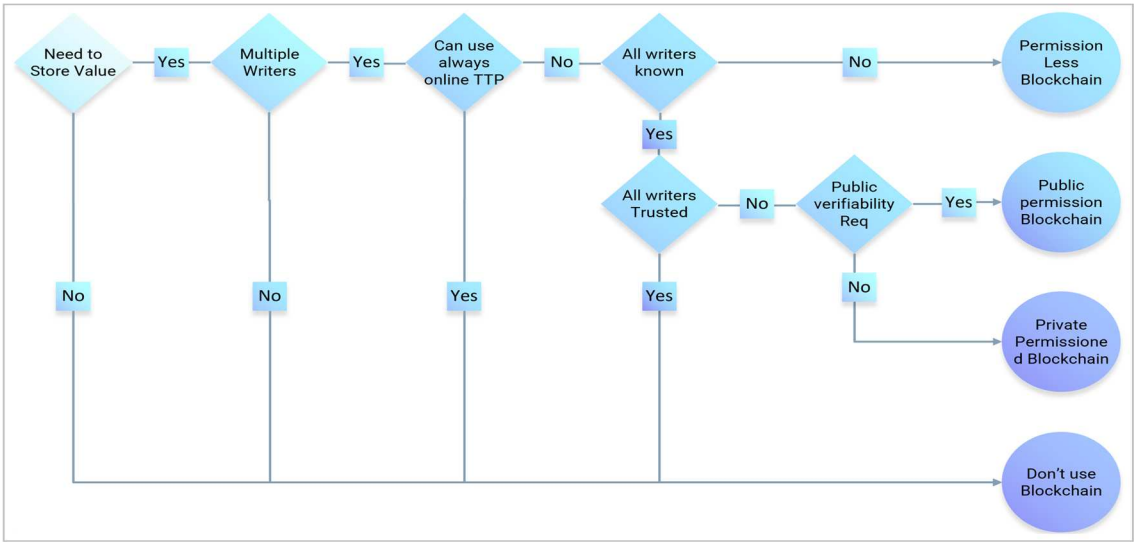

Fig. 2: Selection method for Blockchain [11]

From Table I, e may readily deduce that blockchain currently has insufficient throughput to cope with massive transaction volumes. However, experimenting with this method for protecting HIDS DBs is beneficial.

V. FRAMEWORK FOR SELF HEALING INTRUSION DETECTION

Based on the discussion above, we propose a novel framework for a Self-Healing Intrusion

Detection System based on file integrity checkers (Fig. 3). The proposed framework is envisaged to provide protection against attacks at Section III. This framework will use Blockchain to store system state to ensure integrity of protected files. Broadly the proposed framework will work as per following contours:

1) Proposed framework to be preferably implemented in Linux (can be any OS). Linux kernel can be extended using Stackable File System, Custom System Calls and Linux Security Module framework.
2) Proposed framework will be used to detect changes in critical files and malicious updates. Proposed implementation will be suitable for a server system.
3) DBs for file protection be constructed on a fresh system as per desired policy.
4) Files selected for protection will be hashed together to get a Merkle root value. Merkel root value will be mined on Blockchain to protect against modification. On every startup, fresh Merkel root value will be compared with value from Blockchain. System will operate if value is same, otherwise files are restored.
5) On every startup, fresh Merkel root value will be com- pared with value from Blockchain. System will operate if value is same, otherwise files are restored.
6) Due to scalability issues, less files be selected for Merkle value which may include kernel with IDS module, important DBs and selected files.
7) Merkle value will be non-persistent. An API is used to communicate between Blockchain and proposed IDS.

VI. RELATED WORK

A somewhat similar approach for file integrity protection has been adopted by authors in [13]. They have used Blockchain for monitoring of files' integrity stored on Cloud infrastructure. This approach differs from our proposed framework in following respects:

- Our approach is designed for Intrusion Detection in systems using the File Integrity Monitoring mechanism. The design is provided with self-healing capability through use of Blockchain. However, authors in [13] propose to enable integrity checking of files stored on cloud infrastructure through Blockchain smart contracts instead of Intrusion Detection.
- Our design proposes to use Blockchain smart contract for storage of file checksum or fingerprint only. Whereas authors in [13] propose to use smart contract for storage of files as well as their integrity checks through 3rd parties.
- Our design proposes to use fingerprint of single file or an entire DB for intrusion protection of a system. In contrast, authors in [13] use integrity data at a single file level for integrity monitoring.
- Our design is envisaged to cater for any Intrusion Detection System, which has a DB for monitoring reference. The proposed system in [13] only caters for integrity monitoring of files stored on cloud infrastructure.

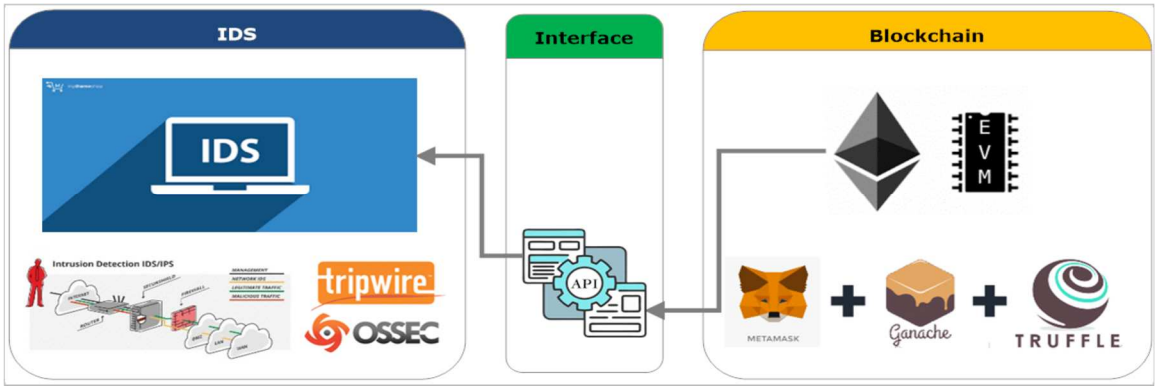

Fig. 3: Proposed IDS Design - Functional Layers

VII. CONCLUSION

In this paper, an endeavor has been made to analyse various types of file integrity checking HIDs and their attacks along with few protection mechanisms. We have also presented a novel approach of a self-healing HIDS. The proposed approach uses Blockchain for securing the HIDS by storing its DB checksum. It is envisaged that the proposed solution can provide suitable protection against all identified threats mentioned in Section III. In future, we intend formulating detailed

structure of the proposed framework along with a viability test for later development.

VIII. REFERENCES

[1] V. Bukac, P. Tucek, and M. Deutsch, "Advances and challenges in stan- dalone host-based intrusion detection systems," in *International Con- ference on Trust, Privacy and Security in Digital Business*. Springer, 2012, pp. 105–117.

[2] S. Patil, A. Kashyap, G. Sivathanu, and E. Zadok, "I3fs: An in-kernel integrity checker and intrusion detection file system." in *LISA*, vol. 4, no. 1, 2004, pp. 67–78.

[3] D. Mosley, "Implementation of a file integrity check system," *East Carolina University*, 2006.

[4] Y.M. Motara and B. Irwin, "File integrity checkers: State of the art and best practices," 2005.

[5] W. Arbaugh, G. Ballintijn, and L. Van Doorn, "Signed executables for linux," Tech. Report CS-TR-4259. University of Maryland, Tech. Rep., 2001.

[6] L. Catuogno and I. Visconti, "An architecture for kernel-level verification of executables at run time," *The Computer Journal*, vol. 47, no. 5, pp. 511–526, 2004.

[7] G. Sivathanu, C. P. Wright, and E. Zadok, "Ensuring data integrity in storage: Techniques and applications," in *Proceedings of the 2005 ACM workshop on Storage security and survivability*, 2005, pp. 26–36.

[8] W. Arbaugh, G. Ballintijn, and L. Van Doorn. (2001) Do you need a blockchain? [Online]. Available: https://drum.lib.umd.edu/handle/1903/1139

[9] A. Chuvakin, "Ups and dows of unix/linux host-bases security solu- tions," *; login:: the magazine of USENIX & SAGE*, vol. 28, no. 2, pp. 57–62, 2003.

[10] D. Armstrong. (2003) An introduction to file in- tegrity checking on unix systems. [Online]. Avail- able: https://www.giac.org/paper/gcux/188/introduction-file-integrity- checking-unix-systems/104739

[11] A. G. Karl Wüst. (2017) Do you need a blockchain? [Online]. Available: https://eprint.iacr.org/2017/375.pdf

[12] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[13] A. Pinheiro, E. D. Canedo, R. T. De Sousa, and R. D. O. Albuquerque, "Monitoring file integrity using blockchain and smart contracts," *IEEE Access*, vol. 8, pp. 198 548–198 579, 2020.