International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 13, No. 2, July 2024, pp. 395~402
ISSN: 2089-4864, DOI: 10.11591/ijres.v13.i2.pp395-402

Kernel rootkit prevention model using multiclass

Suresh Kumar Srinivasan, Sudalaimuthu Thalavaipillai
Department of Computer Science and Engineering, Hindustan Institute of Technology and Science, Chennai, India

Article Info

Article history:

Received Apr 23, 2023
Revised Sep 1, 2023
Accepted Sep 17, 2023

Keywords:

Cloud
File
Malware
Message diggest 5
Process
Rootkit
SHA-256

ABSTRACT

Malicious individuals can access a computer network or application thanks to a series of programmes known as rootkit malware. These kernel rootkits use covert methods to conceal the kernel components, various control frameworks, and system activities, making identifying or prohibiting their presence in the target machine challenging. The bulk of rootkit detection and prevention techniques used today are particular to a system and dependent on recognized sources, making them ineffective for growing, evolving, concealed, or unnamed rootkits. This study proposes using the kernel rootkit prevention model using multiclass (KRPMM) system to identify hash values and detect/prevent such rootkits. The file downloaded by the client, who is availing of the service, is not permitted into the node used by the client in the cloud. But, it is redirected to the node wherein the file that has been downloaded and has entered the node anew is examined by a program which is specially coded to test the presence of rootkit in the file by some mechanisms and then comes to a conclusion of either the file being malicious or the file being clean and is free of rootkits. KRPMM tested only 64 rootkits.

Corresponding Author:

Suresh Kumar Srinivasan
Department of Computer Science and Engineering, Hindustan Institute of Technology and Science
1 Rajiv Gandhi Salai (OMR) Padur, Chennai-603103, Tamil Nadu, India
Email: sureshkumarphd2018@gmail.com

1. INTRODUCTION

A VMware system is an information technology solution that creates and manages computer instances virtual machines (VMs) using VMware's virtualization technologies [1]. Because virtual systems allow for the abstraction and distribution of physical hardware components like the central processing unit, memory, storage, and the internet they can be flexible, effective, and scalable in a VMware context. Since each VM has its operating system, set of apps, and configuration options, it functions like a separate physical machine. VMware setups are a potent tool for rootkit identification because they can be used to build isolated, controlled virtual environments that can be altered without impacting the host operating system. As a result, malware and rootkits functioning in the virtual environment can be analyzed and observed without affecting the underlying operating system or other systems on the network.

Kernel rootkits [2] typically intercept system calls sent to the operating system by user-mode programs and change the responses that these calls return. This may enable a rootkit to conceal files, programs, network links, and other system activity from detection. A kernel rootkit in a VMware environment can compromise the virtualization layer, significantly affecting the overall system's security. Kernel rootkits can also intercept network traffic and plug into device drivers to mask their presence and activities further. Many approaches have been put forth to identify kernel rootkits. The two main kinds of these methods are static methods and dynamic methods. The static techniques collect the distinguishing characteristics for dividing up excellent and bad kernel modules using static analysis. It will be challenging for static analysis tools to obtain the logical properties of the kernel modules if they use complicate techniques. It is advised to use the dynamic strategy to solve the hiding data issue.

The fundamental tenet of this strategy is to run a kernel module in an appropriate setting before observing its behavior in real-time and making a decision subsequently. Most currently used techniques use emulation (such as quick emulator (QEMU)) [3] to provide the kernel modules' execution environment. Certain malicious kernel components can identify a VM that is in use before changing their behaviour, and because some kernel modules may depend on particular system components that the software does not support, they might not function properly there to address the above limitation. These drawbacks are caused by the emulation, which adds a significant performance penalty. In this work, employing the SHA-256 algorithm aids efficiently and effectively in rootkit detection in a virtual system [4]. Analysing each hashing algorithm's unique characteristics and the needs of the use case is essential when choosing one, such as message diggest 5 (MD5) and SHA-256. Here are some of the reasons why MD5 and SHA-256 may be used. MD5, efficiency, legacy compatibility, and non-security applications are all advantages of using MD5. Strong security, cryptographic standards, and long-term security are all terms used to describe SHA-256.

The program initially recognizes any records with growth, such as ".exe" records, which are now present in the specified index and are considered executable records. The program first secures the target record's MD5 hash. The program is granted access to a database of recently discovered records, including rootkit or other dangerous software. The MD5 hash estimate of the most recent executable archive obtained is then sent to the program. The SHA-256 computation then hashes the report into its pseudo-random imprint hash, which is similarly distinguished, and the database that contains sections of SHA-256 hashes of different rootkit records as well as the hashes of malware that have recently been seen to be malware or rootkits independently.

The assessment of the execution of deep learning and artificial intelligence (AI) algorithms was performed using memory data. The successful malware detection was attributed to a memory study conducted with a logistic regression analysis method. As a result, conflicting results may occur if a different type of information is used or if the number of different features is unique and cannot be applied to alternative datasets. Data from memory analyses enabled high rates of malware identification accuracy. It has been observed that MapReduce on disk is capable of processing data faster than apache spark [5]. A cloud-based virus detection method, visualized memory change area dimensionality reduction (VMCADR), was developed to ensure uncompromised client privacy. This method immediately detects malware within images of binary storage without accessing user data. The memory changed area (MCA) files are created using the memory difference (MDIFF) approach with the memory sample, although they require more time to obtain compared to typical files. This technique serves to protect the user's private information [6].

A monitoring tool uses out of VM introspection to check for hooked system calls on the VM and changes to the kernel address space. VM introspection library (LibVMI), handle VM introspection and VM control is dealt with by the libvirt application programming interface (API). It detects a rootkit assault by keeping the contents of a system call function's hash values and contrasting them with hash values routinely produced from live memory. If the scan results are accurate, it sends the user an email and allows them to move the virtual computer to a different system for further forensic analysis. Kernel check scan checks the legitimacy of the kernel address space. The hash values are computed using the MD5 hash method. The simple mail transfer protocol (SMTP) library is then used to send the mail notification to a Gmail SMTP server using transport layer security (TLS). The message is created using multipurpose internet mail extensions (MIME) multipart. The software is only compatible with Linux distributions [7].

A herd of connected VMs has been used to find the kernel-resident malware. The fingerprints are compared, and those that indicate anomalous hosts are found without kernel-specific semantics. They implemented their method into fluorescence and demonstrated that it can identify Linux and Windows hosts infected with kernel-resident malware in real-world situations. In an hour, fluorescent can analyze many VMs running Linux distributions. The standardized values of a kernel's code pages are expressed as a collection of hashes that make up each fingerprint. These hashes were created using fuzzy hashing, which allows comparable page contents to map to similar hash values. Fluorescence was used to identify all contaminated hosts in the Linux and Windows-based groups. It gathers each VM's most recent fingerprint, does feature alignment, and detects anomalies in the data using deep learning techniques. For quicker analysis, large herds can be split up into smaller groups. Without the necessity for training over particular ones, it employs virtual server introspection-based insights to find abnormalities. Only a portion of the kernels under investigation are known. Code that has been just-in-time (JIT)-compiled needs to be taken into account [8].

Cloud-based malware detection and mitigation system have been presented, and it has relied on the signatures like MD5 and SHA1 and the characteristics of many different families of contemporary malware. In this study, cloud services deliver cloud-based antivirus, and intrusion detection systems will use hash signatures and patterns. Additionally, they suggest a method for cloud-based real-time malware detection and prevention. The signatures are downloaded from various sites, including virus total, utilizing a hash calculator to generate hashes in SHA and MD5. The system will compare any folder or directory being scanned with the pre-existing signatures from the database. The file is classified as unknown if the signature cannot be found, and rule-based detection is then performed. If any string satisfies the rule, the file is considered malicious, an alert is generated, and the database is updated with the signature of the new file. Cloud security was enhanced by real-time malware analysis. Property signature-based systems can achieve very high detection speeds due to their small size precision and low number of false positives. Signatures can only be created after identifying a malware sample [9].

A unique rootkit detection method adapted to the cuckoo sandbox is based on the idea of aggregating and trending micro locality sensitive hashing (TLSH), and it aims to give interactive analytical reports on documents by analysing them in a safe setting. Leveraging chi-square, random forest, and principal component analysis (PCA) strategies, the most crucial traits are chosen. To construct clusters, the model compares the hash values of various files and sets a threshold limit. The clustering strategy using TLSH might provide a robust intrusion detection systems approach without compromising performance, and the predictive accuracy demonstrates the upgraded version of the classifier and a decreased amount of false positives. The scalability of the technique is limited and requires several hours to cluster a set of several hundred samples [10].

Using the WEKA classifier and random forests with a Jupyter Notebook helps identify and categorize malware dynamically. The cuckoo sandbox and XEN cloud platform were the foundation for the cloud test. They monitored various machine learning (ML)-based classifiers to increase the effectiveness of malware analysis. Relying upon the attribute selection derived by the WEKA, the information gain ratio feature extraction would be capable of achieving the most pertinent traits that optimize the reliability of the information available. The processing time required to create the model depends on the dataset size [11].

This work developed the trusted kernel rootkit detection (TKRD) technique for automatically detecting kernel rootkits in VMs from private clouds performed on a kernel-based VM (KVM) hypervisor. Memory dumps from the VM are inspected utilizing a memory forensic analysis approach every ten minutes to discover harmful functionalities. The characteristics of veiled orphan threads, kernel units, callbacks, driver objects, device trees, timers, and the system service descriptor table (SSDT) function are utilised to construct a range of machine-learning classification frameworks in addition to based on regulations classifiers, trees of choice, support vector machines, and Bayesian models. The data capture interval can be changed and flexibly programmed for rootkits that run for short or extended periods as an exchange between efficiency and efficacy. Memory dumps were used to generate the extensive features that precisely identify kernel rootkits and reveal their peculiar activities. It can only be run in the public cloud and rootkit samples are executed one at a time on the VM [12].

To assess whether a client file type is harmful, this study compares the attributes of .exe files. The decision tree separates the nodes according to the variables and then selects the division that yields the most homogenous subnodes. Instead of using memory analysis, heuristics, or analytical detection techniques, and signature-based detection should be emphasized [13]. A malicious dataset to efficiently discover distinguishing traits that offer high-performance malware detection for both known and new threats. After identifying it as cruel or usual, the system stores each file and combines it with ML classifiers. The feature extraction and detection phases comprise the two cloud-based detection sections. When a user delivers a questionable document over the system, the server responds with an analysis indicating whether or not the file is malware. Some parts of the malware variants remain undetected because of the use of sophisticated code obfuscation methods [14].

2. METHOD

The content of this section explains how to get rid of dangerous software such as rootkits. In simple terms, the procedure is that the client using the service is not allowed to upload the content downloaded onto the node (system), and the client is using it in the cloud. The file that has been downloaded and has entered the node anew is scrutinized by a program that is specifically coded to test for the presence of a rootkit in the file using some mechanisms, and it then determines whether the file is malicious or clean and free of rootkits. This dictates whether the recently downloaded file is returned to the node used by the client provided or the file is free of the rootkit [15]. If not, the file is discarded and the information is reported to the client by a message in a text file instead of the downloaded file. The file will not be provided even if the user is willing to download it with its risks. Thus, the file is blocked and makes the node used by the client in the cloud free of rootkits. This is explained in detail about how the mechanism works on two given platforms on any two nodes on the cloud service. This may also render security to systems not part of the cloud services. It may also enhance the security of all machines. Architecture diagram of kernel rootkit prevention model using multiclass (KRPMM) is shown in Figure 1.

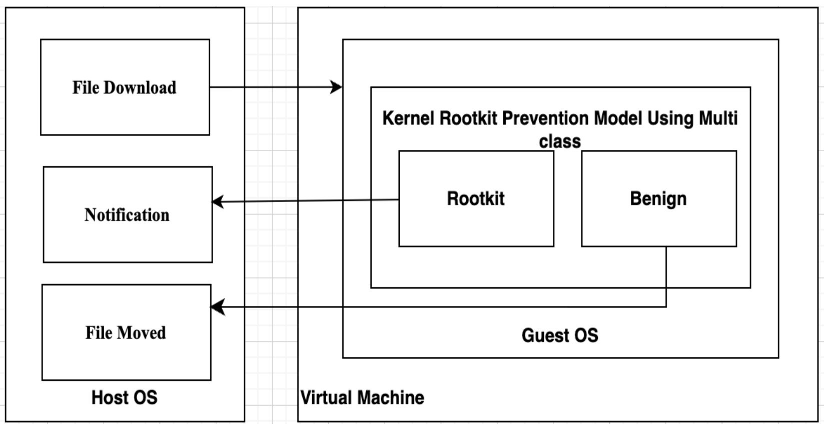

Figure 1. Architecture diagram of KRPMM

The target computer that needs to be secured must first be mapped to the node that contains the rootkit prevention mechanism for the software to run and the rootkit [16] to be prevented. The node's browser is set to the recently mapped disc and links to the other node or the node with the prevention mechanism. This step ensures that the download is skipped to the other node and diverted away from the target node. After that KRPMM is executed. The following step is to generate a text file containing the following information: the file contains rootkit [17] or malicious software that could potentially cause harm or corrupt the system and is deleted without any exceptions.

The program will only send this text file containing the notice above if it determines that the incoming file, which the client received, is malicious or a rootkit [18]. Once the program has been downloaded, the indicated protection measure will be implemented once it is present in the other node to which the file is being transferred. This makes the client node rootkit safe by ensuring that the incoming file cannot infect it [19].

2.1. Rootkit prevention mechanism

As mentioned above, the rootkit prevention mechanism involves the file being bypassed to the node where the prevention program resides. Now, the file has to be processed to have an insight into whether the file is free of any malicious software or possesses a rootkit or malicious software that may threaten the system. A directory is assigned for quarantining the file suspected of having a rootkit or malicious software that threatens the system's security or subverts the whole of the deposit to corrupt the node or exploit the resources of that node and its sensitive data.

2.1.1. Detection of rootkits/malicious software

The program first detects any files with an extension, for instance, ".exe," i.e., an executable file currently in the specified directory. The program then obtains the MD5 hash of the target file first. The program is given access to a database of previously found files containing rootkits or malicious software. This proves effective on most rootkits or malicious software in that consent. The program then gets the MD5 hash value of the newly downloaded executable file. The MD5 hash is then processed to meet the requirements such that it would be normalized to compare with any data from the database that contains the MD5 [20] hash values of a vast set of malware or rootkits or any other malicious software which are defined as malware or rootkits or malicious software by its previous encounters by anti-virus software; thus, updating the database with the "MD5" hashes of that software respectively.

2.1.2. Prevention measures

The program, consisting of the MD5 hash of the executable file to be tested for discrepancies, will be compared with the data values obtained from the database and will be cross-checked for equality. This test reports the crucial result. This result will be one of the major deciding factors with which the program decides whether to send the file back to the client's node or discard the file and send a file-blocked report to the client. The result will be considered with a higher priority by the program, and that data, in turn, will affect the transfer of the executable file. If the executable file contains no such identifiable and significant difference, it will be sent back to the client's node without any modification to the downloaded executable file. When the executable file is clean, the program takes control to send the same executable file that was received by the node from the client's node, which was bypassed to this node, is sent to the client's node by using the drive that was mapped to each other on the network.

Immediately after the MD5 [21] hash algorithm is executed and used for detection, the SHA-256 algorithm is implemented on the same specified file. SHA-256 algorithm then hashes the file into its pseudo-random signature hash, which is also compared with the database that contains entries of SHA-256 hashes of various rootkit files and also the hashes of malware that have been previously found to be malware or rootkits, respectively. Then, the algorithm compares its original file's secure hashing algorithm-256 [22] hash. That is, the original version of the file is sent to a hashing function that implements the SHA-256 hash. The soup "secure hashing algorithm", first converts all bytes to binary format. This then pads the other remaining spaces with 0's. The last few digits are filled with the last of the original message, i.e., the file's previous content at the latest. The padding should be exactly 512 bits of 64 blocks. Separated, the algorithm calculates the hash by interchanging the first half of the 1st word and then performs the suitable shift operation. Then, it serves the soup wherein the algorithm recursively calls the hashing function to the extent that the original message is represented by a pseudo-random string that, in no sense, means the same as the original message.

As the function works, it generates a unique hash for a message, in our case, the file itself. The generated hash is cross-checked for the incoming file. The result will return a Boolean value and the hash values of both files. This is implemented to support the development determined by the MD5 hashing algorithm. Thus, the file downloaded by the client can be tested for any issues regarding rootkits or malicious software without reaching the client's node before it is ensured to be rootkit-free. This also provides the reliability of that file by the client, withholding not a sway against the deceitful nature of the file containing any rootkits.

3. RESULTS AND DISCUSSION

In accordance with the usage of the MD5 and SHA-256 hashing algorithms to track the actions taken by the programme when given a file, whether it's an executable file or a dynamic link library [23]. Different outcomes on the likelihood and promptness of the program to be diagnosed may be obtained by comparing the two algorithms' hashes. The diagnostic file is considered in the form it had when it was first acquired, i.e., without any conversions the program may have made to it or any other changes that may have been reflected on it. Whether it be a rootkit or not, the file is eagerly anticipated. For a more straightforward but effective illustration, consider 64 examples of different rootkit files that are uncategorized, or in other words, not specified to the program that detects and then prevents the rootkit or any other malware from infecting the concerned node. Particular possibilities are formed by the boolean values that the MD5 and SHA-256 [24] algorithms returned. Table 1 (legal and illegal rootkits) illustrates the restriction of rootkit-covered records.

Table 1. Input field

S.no	Malware name	Malware size	Offensive
1	Virus.bat.qwerty.b	676 kb	Ucrtbase.dl
2	Virus.boot.catman	28 kb	Scvhost.exe
3	Virus.unix.sillysh.b	6.84 kb	Aubot.exe
4	9ba7332fdca46ed72bd788def5498140	793 kb	User32.dll
5	38c7bd26550daa3b4527f4eeefe8a0dd	81.5 kb	Svchost.exe
6	D0617fedf0ea31d7d5fb55bd334d85d6	8 kb	Svchost.exe
7	F0f927ee20a62d0b0a1b37d68d1406ea	78 b	Svchost.exe
8	$%&%_2169.vir@	22.86 kb	Taskhost.exe
9	Backdoor.win32.haxdoor.gs	1.26 mb	Taskhost.exe
10	Bakuryu	121 kb	Scvhost.exe
11	Shell.jpg	89.1 kb	Svchost.exe
12	F6e671d8630df5d8045ff4243da94f74	24 kb	Ucrbase.dl
13	Afe8df184dccf6db48cf27916d0d0da6	48 kb	Ucrtbase.dll
14	6eddd98e0463acaa3aa0eeab26b1d3c9	1 kb	Ucrbase.dll
15	80da4801d2b70d7044e9d660a05c676	5.03 kb	Svchost.exe
16	4356aded80ee30d1f85321ecc28694b3	140 b	Taskhost.exe
17	E08de794d84c472b1fd9a862bd729556	107 b	System32.dll
18	Rootkit.win32.agent.agk	512 b	Ucrbase.dll
19	Rootkit.win32.agent.azt	512 b	Ucrbase.dll

Description of Table 2, when done on the optimal launch system in Windows 2022 against 64 rootkits, every test that tries to differentiate the cloud fails. Each test includes the rootkit [25], which attacks a brand-new boot system. No false positives are produced by this system (100% certified negatives, 0 false positives). The ideal Windows instance system is expected to continue to be flawless. 64 rootkits were tested in this study to see if they could be detected as rootkits. In contrast to the remaining 58 rootkits, 42 are included in the validation set. The remaining 16 of the 58 rootkits need to be distinguishable, leaving only 6. This result showed 23.63% false negative and 76.36% positive. An incorrect rootkit installation setup was the root of the false negative. Attackers perfect their rootkit installation techniques, lowering the false negatives rate.

Description of Table 3, when injecting the rootkit into the computer, it affects the file, process, and port. Obtaining the SHA-256 and MD5 file values and contrasting these values with the data in the database. The rootkit can access that specific file if the SHA-256 and MD5 results are true. The detection process time is mentioned in Table 3. This procedure is carried out on both Windows and Ubuntu computers. Table 3 shows that Ubuntu has a faster detection rate than Windows. 8 GB Windows and Ubuntu rootkit detection time show in Figure 2.

Table 2. Confusion matrix of kernel rootkit prevention model multipart

Actual	Predicted (-)	Predicted (+)
-	6	0
+	16	42

True positive: MD5 and SHA 256 are both true.
False negative: MD5 and SHA 256 are both false.
True negative: MD5 is True and SHA 256 false.
False negative: MD5 is False and SHA 256 false.

Table 3. True positive rootkits sample detection time

Rootkit	Detection time in 8 GB volume	
	Windows (ms)	Ubuntu (ms)
Trojan-downloader	222	212
Virus.BAT	220	212
Downloader-RW24	220	213
Artemis!7CD08372064A(exe)	214	212
Artemis!21436A8F3E57	215	211
Downloader-AWM.gen	213	212
Virus.BAT.Qwerty.b	214	212
Artemis!629A4B4ADF6E	213	212
W32Fujacks	215	211
PWS-gamania.gen.a	213	211
BackDoor-DIQ	221	212
Vanquish.dll	221	211
Vanquish.exe	222	212
ASBV	223	213
W-boot	225	213

Accuracy of prevention: there are five evaluation metrics for this model in this area. The rootkit's effectiveness at preventing attacks is the primary metric and is calculated as (1) to (5).

$$Accuracy = \frac{Rootkit\ Tested - True\ Positives - False\ Negatives}{Rootkit\ Tested} \quad (1)$$

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (2)$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (3)$$

$$Fscore = 2x\left(\frac{Precision\ Recall}{P\ recision + Recall}\right) \quad (4)$$

$$G\ mean = \sqrt{Precision * Recall} \quad (5)$$

The detection time lengthens with volume size. The rootkit acknowledgment time is quicker with this system than with other rootkit detection methods like the mark-based approach and the equipment-based localization procedure. Only when the limit volume in this examination stays constant will it be quicker to locate the volumes that need 8 GB. The performance measure can be seen in Figure 3. The accuracy of positive classifications, or the likelihood that an anomalous variation detected by the detector has been correctly classified, is measured by precision. The degree of efficiency in the scanner's identification of any freshly tested information set. Recall measures the detector's ability to detect variations or the likelihood that an aberrant examples of will be appropriately classified as such. By partially accounting for all the data, the final two measures, F score and G mean, offer a more full assessment of a particular detector's performance.

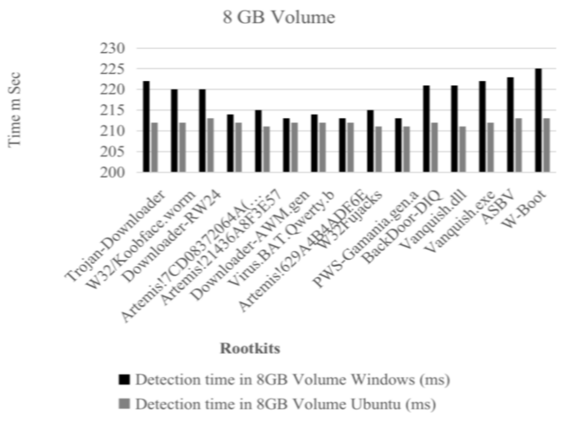

Figure 2. 8 GB Windows and Ubuntu rootkit detection time

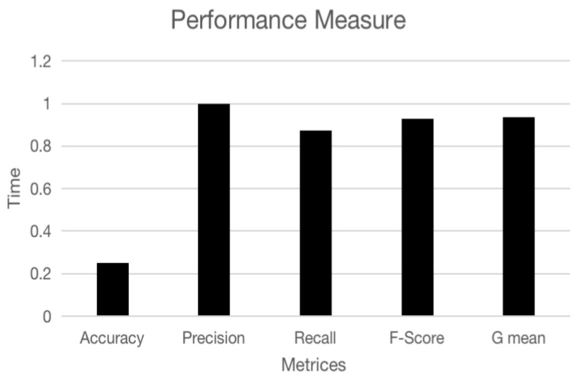

Figure 3. Performance measure

4. CONCLUSION

This study sheds light on the present state of rootkit technology in comparison to novel detection techniques. The KRPMM rootkit was discussed in general terms as well as in technical detail. The primary methods for combating rootkits were examined, and their drawbacks were emphasised. The program then first secures the target record's MD5 hash. The program is granted access to a database of recently discovered papers that include rootkit or other dangerous software. The MD5 hash estimate of the most recent executable archive obtained is then sent to the program. The SHA-256 computation then hashes the report into its pseudo-random imprint hash, which is similarly distinguished, and the database that contains sections of SHA-256 hashes of different rootkit records as well as the hashes of malware that have recently been seen to be malware or rootkits independently. Each method for confirming the kernel rootkit avoidance model that was mentioned above, the main problem of multiclass rootkits is the requirement for proper configuration on various system. There is now a demand for additional rootkit detection techniques that are simple to configure and do not significantly impact performance.

ACKNOWLEDGEMENTS

The authors would like to thank the Department of Computer Science and Engineering, Hindustan Institute of Technology and Science, Chennai, India, for providing the infrastructure to develop the models for the research problem.

REFERENCES

[1] L. F. Ilca, O. P. Lucian, and T. C. Balan, “Enhancing cyber-resilience for small and medium-sized organizations with prescriptive malware analysis, detection and response,” *Sensors*, vol. 23, no. 15, p. 6757, Jul. 2023, doi: 10.3390/s23156757.
[2] D. Tian, Q. Ying, X. Jia, R. Ma, C. Hu, and W. Liu, “MDCHD: A novel malware detection method in cloud using hardware trace and deep learning,” *Computer Networks*, vol. 198, pp. 1–10, Oct. 2021, doi: 10.1016/j.comnet.2021.108394.
[3] A. Tsohou, V. Diamantopoulou, S. Gritzalis, and C. Lambrinoudakis, “Cyber insurance: state of the art, trends, and future directions,” *International Journal of Information Security*, vol. 22, no. 3, pp. 737–748, 2023, doi: 10.1007/s10207-023-00660-8.
[4] S. Manoharan, P. Sugumaran, and K. Kumar, “Multichannel based IoT malware detection system using system calls and opcode sequences,” *International Arab Journal of Information Technology*, vol. 19, no. 2, pp. 261–271, 2022, doi: 10.34028/iajit/19/2/13.
[5] M. Shobana and S. Poonkuzhali, “An efficient botnet detection approach for green IoT devices using machine learning techniques,” *Journal of Green Engineering*, vol. 10, no. 3, pp. 1053–1076, 2020.
[6] K. R. Sowmia and S. Poonkuzhali, “Artificial intelligence in the field of education: a systematic study of artificial intelligence impact on safe teaching learning process with digital technology,” *Journal of Green Engineering*, vol. 10, no. 4, pp. 1566–1583, 2020.
[7] B. U. A. Barathi and S. Poonkuzhali, “Design and implementation of interactive data analytics model for predicting the survivability of breast cancer patients,” *Journal of Environmental Protection and Ecology*, vol. 21, no. 4, pp. 1455–1468, 2020.
[8] R. Vadivel and T. Sudalaimuthu, “Cauchy particle swarm optimization (CPSO) based migrations of tasks in a virtual machine,” *Wireless Personal Communications*, vol. 127, no. 3, pp. 2229–2246, 2022, doi: 10.1007/s11277-021-08784-7.

[9] J. Jeyalakshmi and S. Poonkuzhali, "Prescriptive analytics of constraint optimisation of diabetes diet exhortation by using information systems," *Journal of Environmental Protection and Ecology*, vol. 22, no. 6, pp. 2672–2681, 2021.

[10] I. Tariq, M. Nazish, S. Ashaq, I. Sultan, and M. T. Banday, "A performance comparison of hashed and authenticated advanced encryption standard," *Proceedings - 2nd International Conference on Smart Technologies, Communication and Robotics 2022, STCR 2022*, pp. 1–5, 2022, doi: 10.1109/STCR55312.2022.10009112.

[11] S. Poonkuzhali, J. Jeyalakshmi, "Study of diabetes mellitus patients for thyroid related co-morbidities using data analytics," *Basic and Clinical Pharmacology and Toxicology*, vol. 124, no. S3, pp. 19–20, 2019,

[12] C.-L. Chen and S. Punya, "An enhanced WPA2/PSK for preventing authentication cracking," *International Journal of Informatics and Communication Technology (IJ-ICT)*, vol. 10, no. 2, pp. 85–92, Aug. 2021, doi: 10.11591/ijict.v10i2.pp85-92.

[13] A. Vijayaraj, R. M. Suresh, and S. Poonkuzhali, "Node discovery with development of routing tree in wireless networks," *Cluster Computing*, vol. 22, pp. 10861–10871, 2019, doi: 10.1007/s10586-017-1211-y.

[14] A. Vijayaraj, R. M. Suresh, and S. Poonkuzhali, "Load balancing in wireless networks using reputation-ReDS in the magnified distributed hash table," *Multimedia Tools and Applications*, vol. 77, no. 8, pp. 10347–10364, 2018, doi: 10.1007/s11042-018-5620-6.

[15] P. Sugumaran, K. K. Ravi, and T. Shanmugam, "A novel algorithm for enhancing search results by detecting dissimilar patterns based on correlation method," *International Arab Journal of Information Technology*, vol. 14, no. 1, pp. 60–69, 2017.

[16] Q. Wang and Q. Qian, "Malicious code classification based on opcode sequences and textCNN network," *Journal of Information Security and Applications*, vol. 67, 2022, doi: 10.1016/j.jisa.2022.103151.

[17] N. Aman, Y. Saleem, F. H. Abbasi, and F. Shahzad, "A hybrid approach for malware family classification," *Communications in Computer and Information Science*, vol. 719, pp. 169–180, 2017, doi: 10.1007/978-981-10-5421-1_14.

[18] B. Liu *et al.*, "An approach based on the improved SVM algorithm for identifying malware in network traffic," *Security and Communication Networks*, vol. 2021, pp. 1–14, Apr. 2021, doi: 10.1155/2021/5518909.

[19] A. Ullah, I. Laassar, C. B. Şahin, O. B. Dinle, and H. Aznaoui, "Cloud and internet-of-things secure integration along with security concerns," *International Journal of Informatics and Communication Technology (IJ-ICT)*, vol. 12, no. 1, pp. 62-71, Apr. 2023, doi: 10.11591/ijict.v12i1.pp62-71.

[20] M. Awais, Q. Abbas, S. Tariq, and S. H. Warraich, "Blockchain based secure energy marketplace scheme to motivate P2P microgrids," *International Journal of Informatics and Communication Technology (IJ-ICT)*, vol. 11, no. 3, pp. 177–184, Dec. 2022, doi: 10.11591/ijict.v11i3.pp177-184.

[21] X. Wang, J. Zhang, A. Zhang, and J. Ren, "TKRD: Trusted kernel rootkit detection for cybersecurity of VMs based on machine learning and memory forensic analysis," *Mathematical Biosciences and Engineering*, vol. 16, no. 4, pp. 2650–2667, 2019, doi: 10.3934/mbe.2019132.

[22] B. Yergaliyeva, Y. Seitkulov, D. Satybaldina, and R. Ospanov, "On some methods of storing data in the cloud for a given time," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 20, no. 2, pp. 366–372, Apr. 2022, doi: 10.12928/TELKOMNIKA.v20i2.21887.

[23] H. S. Hamid, B. AlKindy, A. H. Abbas, and W. B. Al-Kendi, "An intelligent strabismus detection method based on convolution neural network," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 20, no. 6, pp. 1288–1296, Dec. 2022, doi: 10.12928/TELKOMNIKA.v20i6.24232.

[24] S. M. Kareem and A. M. S. Rahma, "A new multi-level key block cypher based on the Blowfish algorithm," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 18, no. 2, pp. 685–694, Apr. 2020, doi: 10.12928/TELKOMNIKA.V18I2.13556.

[25] I. Ahmed, "Technology organization environment framework in cloud computing," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 18, no. 2, pp. 716–725, Apr. 2020, doi: 10.12928/TELKOMNIKA.v18i2.13871.

BIOGRAPHIES OF AUTHORS

Suresh Kumar Srinivasan is a research scholar at the Hindustan Institute of Technology and Science in Chennai, India's Department of Computer Science and Engineering. The University of Madras in Chennai awarded him a bachelor of engineering in computer science and engineering, and Anna University in Chennai awarded him a master of engineering in computer science and engineering. He current research is focused on cloud computing security. He is a CSI permanent member. He can be contacted at email: sureshkumarphd2018@gmail.com.

Prof. Sudalaimuthu Thalavaipillai works at the Hindustan Institute of Technology and Science in Chennai, India, in the School of Computing Science. He received his Ph.D. from the Hindustan Institute of Technology and Science in Chennai, India. He is certified as an ethical hacker. In respected international journals and conferences, he has 50 research articles published. Both Australia and India grant him patents. He received many awards throughout his career, including the Top Innovator Award and the Pearson Award for Best Teacher. His areas of interest in the study include cyber network security, grid and cloud computing, and machine learning. He has CSI, IEEE, and ACM lifetime memberships. He can be contacted at email: sudalaimuthut@gmail.com.