Deception-based Method for Ransomware Detection

TaeGuen Kim[1*]

[1*] Department of Information Security, Soonchunhyang University, Korea. tg.kim@sch.ac.kr,
Orcid: https://orcid.org/ 0000-0002-6586-2037

Received: June 22, 2023; Accepted: August 23, 2023; Published: August 30, 2023

Abstract

Ransomware is a rapidly growing malware threat that encrypts a user's files and demands a ransom for the decryption key. It has caused significant financial harm worldwide and is difficult to detect, especially when it's a new, unknown zero-day ransomware. Most commercial antivirus software relies on signature-based detection, which can be slow and inadequate for swiftly identifying suspicious programs. To tackle these challenges, this paper presents a ransomware protection method utilizing decoy files. Our deception-based protection method enhances ransomware detection with a fair decoy deployment strategy. Our method offers the advantage of robustly detecting ransomware compared to existing deception-based methods. Furthermore, it can effectively address ransomware that employs random access attacks, thereby bypassing deception-based detection techniques. In the evaluation, we provide a comprehensive analysis of our experimental results to vividly demonstrate the efficacy of our proposed method. Specifically, we introduce a random-access attack scenario that could potentially circumvent deception-based protection mechanisms. Furthermore, we assess the resilience of our method against such random-access attacks.

Keywords: Deception-based Method, Malware Detection, Ransomware Detection.

1 Introduction

Ransomware attacks targeting Windows-based systems has been observed globally, with a corresponding rise in the victim count. A report from Secure List (Securelist, 2021) indicated that in 2020, around 1,091,454 unique users experienced ransomware threats on their devices. Another report by Trend Micro (Trend Micro, 2021) highlighted that in May 2017 alone, WannaCry infected over 200,000 individuals across 150 nations. While ransomware shares several characteristics with typical malware, it possesses a distinct trait. Ransomware performs the file operations swiftly to encrypt users' system files. Moreover, once a system is compromised and files get encrypted, decrypting them without the necessary keys becomes a very difficult task (Ferreira, A.P., 2021). Even with comprehensive manual techniques like reverse engineering to unveil the ransomware's actions, the decryption keys might remain elusive, especially if they are to be transmitted from a remote server. Hence, when a system gets infiltrated by ransomware, the prospects of restoring it without the perpetrator's intervention diminish considerably.

Historically, antivirus solutions have relied on signature-based detection techniques to identify malware (Al-Asli, M., 2019). These methods use static patterns, sourced from recognized malicious binaries, to check if unfamiliar binaries are malicious. The patterns, in signature-based detections,

Journal of Internert Services and Information Security (JISIS), volume: 13, number: 3 (August), pp. 191-201
DOI: *10.58346/JISIS.2023.I3.012*

*Corresponding author: Department of Information Security, Soonchunhyang University, Korea.

encompass strings and features of disassembled codes. While these methods are good at finding malware variations, they struggle with new, unknown threats because the patterns for such malware haven't been found yet. Signature-based methods may not detect malware types that change quickly and produce many versions (Nieuwenhuizen, D., 2017). Even though some methods use patterns from complex dynamic analysis results, most antivirus software in a user system cannot perform dynamic analysis because it takes a lot of computing power (Bayer, U., 2006). To overcome the limitation, many ransomware detection approaches have been proposed. The existing approaches can be divided into two methods: deception-based method and behavior-based method. Deception-based methods use fake files to detect when these files are accessed, while behavior-based ones watch file activities to see if anything unusual is happening. We are primarily dedicated to the development of a novel deception-based ransomware detection method that offers deterministic ransomware detection among various available approaches. Unlike previous deception-based methods, which struggle with random-access attacks caused by decoy files closely located in the root directory, our proposed method can effectively detect ransomware that employs random access attacks with only a few decoy files distributed across the user subfile systems. In order to efficiently combat ransomware that employs random access attacks, our methods employ the spectral clustering algorithm to evenly distribute decoy files.

2 Related Works

Many previous research has delved into understanding the operations of ransomware within a victim's system. Daniel et al. (Gonzalez, D., 2017) explored the behaviors of prevalent ransomwares and assessed various preventive measures. Gazet (Gazet, A., 2010) Conducted research on early-emerging ransomware samples to decipher their behaviors. They employed reverse engineering to scrutinize fifteen samples, focusing on malware features and cryptographic elements. Pathak et al. (Pathak, P.B., 2016) examined several ransomware samples, detailing the distinct behaviors of each. They also put forward preventive measures, like regular file backups and timely updates of security tools. Cabaj et al. (Cabaj, K., 2015) executed an in-depth study on CryptoWall, revealing that it communicates via the HTTP protocol and employs concealment tactics, including Tor and misleading DNS.

To shed light on ransomware characteristics, numerous studies proposing protection strategies have been introduced. Jung et al. (Jung, S., 2018), along with Lee et al. (Lee, K., 2019), put forward models to assess the entropy in file binary content, ascertaining if a file system's files are encrypted. Scaife et al. (Scaife, N., 2016) introduced a behavior-oriented strategy for defense against ransomware, utilizing specific indicators to identify any resemblance to ransomware activities. Kharaz et al. (Kharaz, A., 2016) unveiled a system, UNVEIL, that dynamically detects ransomware by monitoring I/O actions and evaluating data entropy. Moreover, it uses screenshots to match against known ransomware visuals. Continella et al. (Continella, A., 2016) crafted ShieldFS, aiming to safeguard Windows systems. Al-Rimy et al. (Al-Rimy, B.A.S., 2020) unveiled a technique for spotting ransomware's encrypting patterns. Methods presented in (Jung, S., 2018) (Lee, K., 2019) (Scaife, N., 2016) (Kharaz, A., 2016) (Continella, A., 2016) (Al-Rimy, B.A.S., 2020) recognize ransomware intrusions based on extensive system alterations due to file encryption. Our method stands out by rapidly detecting ransomware through strategic decoy files, apt for real-time system monitoring.

Beyond behavior-centric strategies, deception-focused approaches have also been explored. Chakkaravarthy et al. (Chakkaravarthy, S.S., 2020) proposed a strategy for identifying ransomware in IoT contexts, monitoring factors like CPU load and memory consumption. Feng et al. (Feng, Y., 2017) advocated for a decoy file-centric framework, while Moore (Moore, C., 2016) delved into creating honeypots. Kok et al. (Kok, S., 2019) presented comprehensive ransomware analysis, discussing pros

and cons of contemporary techniques. In contrast, our research introduces an enhanced decoy distribution strategy, boosting resistance to random access attacks.

There are some approaches utilize machine-learning techniques for ransomware detection. Takeuchiet al. (Takeuchi, Y., 2018) centered on n-grams from API sequences, applying them to SVM classifiers. Poudyal et al. (Poudyal, S., 2018) introduced a model focusing on assembly-level attributes. Various algorithms, from decision trees to random forests, were deployed for classification tasks. Maniath et al. (Maniath, S., 2017) implemented an LSTM neural network. Cusack et al. proposed a method that analyzes network traffic characteristics for detection. Subsequent studies (Poudyal, S., 2019) (Almashhadani, A.O., 2019) (Chen, Z.G., 2017) (Sharmeen, S., 2020) (Bae, S.I., 2020) employed diverse machine-learning algorithms, from deep learning to SVM, for effective detection. However, these (Takeuchi, Y., 2018) (Poudyal, S., 2018) (Maniath, S., 2017) (Cusack, G., 2018) (Poudyal, S., 2019) (Almashhadani, A.O., 2019) (Chen, Z.G., 2017) (Sharmeen, S., 2020) (Bae, S.I., 2020) typically assume a sandboxed environment, ensuring ample analysis time and bypassing potential system overheads.

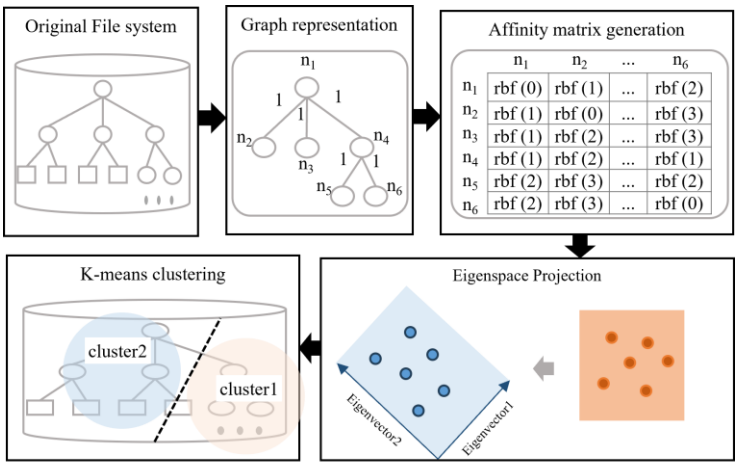

Figure 1: Process Flow of File System Partitioning

3 Proposed Method

Our deception-based protection method is devised to ensure the even distribution of decoy files across the entire file system while preventing their clustering in specific, closely located areas. Initially, the method employs a graph clustering algorithm to evenly partition the entire file system tree into multiple subfile systems. Subsequently, directories within each subfile system are strategically chosen for deploying the decoy files, considering ransomware access patterns. Once the locations for each decoy file are determined, they are generated and distributed accordingly. Each deployed decoy file is actively monitored to detect any manipulation. The key steps of the deception-based protection module are elaborated in the following subsections.

File System Partitioning Module

To partition the file system, the module employes the spectral clustering method (Chen, B., 2014). Spectral clustering is a technique rooted in graph theory, where it identifies groups of nodes in a graph based on their interconnectedness via edges. Through this graph clustering process, it is possible to derive subfile systems from the overall file system structure.

Figure 1 illustrates the process of file system clustering using the spectral clustering algorithm. Initially, a tree-shaped file system is represented in a fundamental graph format. In this representation, each node within the graph corresponds to a directory in the file system, and the presence of an edge between two nodes signifies direct accessibility between those nodes. From the graph representation of the file system, the module constructs an affinity matrix, often referred to as a similarity matrix. This matrix is instrumental in quantifying the degree of similarity between every pair of nodes in the graph. Within the matrix, both the column and row indices correspond to individual nodes, and each matrix element contains a value indicating the similarity between the specific nodes specified by their respective row and column indices.

The primary objective of graph clustering is to divide the file system into subgroups of directories that exhibit similar spatial characteristics. To assess this similarity, the concept of hopping distance is utilized as a metric. After generating all the hopping distances, we employ a radial basis function to normalize these distances, transforming them into a uniform range between zero and one. This normalization process ensures that all similarities are placed within the same feature space. The calculation of similarity is presented in Equation (1-2).

$$hopdist(\mathrm{a},\mathrm{b}) = h(a) + h(b) - 2 * c(a,b) \quad (1)$$

$$S(x_1, x_2) = \exp\left(-\gamma \times hopdist(x_1, x_2)\right) \quad (2)$$

The function hopdist () computes the hopping distance between two paths and is defined by Equation (1), where a and b denote the previous and current paths, respectively. The similarity calculation, as expressed in Equation (2), involves the variables x_1 and x_2, which denote directory paths. Additionally, γ is a gamma value that signifies the inverse of the standard deviation of the radial basis function.

Once the affinity matrix generation is finalized, the next step involves the computation of eigenvectors and eigenvalues derived from this affinity matrix. These eigenvectors and eigenvalues are employed to embed each instance into the eigenspace, facilitating a more distinct separation of instances. Subsequently, the instances projected onto the eigenspace are subjected to clustering using the k-means algorithm (Mohamad, I.B., 2013). This process enables the generation of clusters for the entire file system.

Decoy File Generation and Deployment

After the entire file system has been partitioned, the deployment of decoy files takes place within specific directories. Ransomware often employs depth-first or breadth-first searches to explore the file system. To account for these behaviors, the decoy files are strategically deployed based on the priority order of directories within the DFS or BFS traversal. Therefore, within each partition, directories with either first or middle priority along the DFS or BFS paths are selected. In cases where two directories with first priority are nested, one is chosen at random.

Algorithm 1 outlines the procedure for identifying these priority vertices within the partitions containing the sub-graph vertices. When generating decoy files, the decoy checker ensures that these files closely resemble the user's real files. This is crucial because the protection mechanism can be circumvented if ransomware detects poorly crafted decoy files. To achieve this, the file type of the decoy file matches the most common file type within the directory it belongs to. The size of the decoy file is set to match the average size of files of the same type within the directory.

Additionally, the decoy file can be populated with either randomly generated values or bytes from a selected real file. If actual file content is used, the file must be truncated to fit the specified size of the decoy file. If the average size of a directory is excessive, the decoy file sizes are constrained based on user settings.

Algorithm 1: Prior Vertex Identification Procedure

```
1:  procedure identify_prior_vertexes(partitions_vset_list)
2:     first_vetexes ← { }
3:     middel_vetexes ← { }
4:     for all partition in partitions_vset
5:        // get heights
6:        min_height ← get_min_height(partition)
7:        max_height ← get_max_height(partition)
8:        median_height = (min_height + max_height) % 2
9:        // get all prior vertexes
10:       first_vertexes += get_vertexes(partition, min_height)
11:       tmp_vertexes = get_vertexes(partition, median_height)
12:       middle_vertexes += get_first_vertex_A_order(tmp_vertexes)
13:       middle_vertexes += get_median_vertex_A_order(tmp_vertexes)    middle_vertexes +=
14: get_last_vertex_A_order(tmp_vertexes)
15:     tmp_vertexes = get_vertexes(partition, max_height)
16:       first_vertexes += get_first_vertex_A_order(tmp_vertexes)
17:     first_vertexes += get_median_vertex_A_order(tmp_vertexes)
18:     first_vertexes += get_last_vertex_A_order(tmp_vertexes)
19:       for all vertex in first_vertexes
20:        //randomly remove one vertex from child-parent pair
21:        for all child_vertex in vertex
22:        if first_vertexes.exists(child_vertex)
23:              if rand( ) / 2 == 0
24:             first_vertexes.remove(child_vertex)
25:              break
26:                else
27:             first_vertexes.remove(vertex)
28:              break
29:    return first_vertexes, middle_vertexes
```

The decoy file names are randomly selected from a dictionary containing readable words. Lastly, the decoy files are configured as "Superhidden" to remain concealed from the user. Superhidden files are not visible to users browsing the file system through Windows Explorer, and compression tools like WinZip typically exclude them when compressing a directory. It's important to note that ransomware might attempt to enable hidden attributes of a file to avoid detection.

Decoy File Check

Once all the decoy files have been successfully deployed, the decoy checker initiates the monitoring of file input/output (I/O) activities on these decoy files. If a decoy file is accessed or tampered with, the decoy checker promptly takes mitigation measures, which may include alerting users and terminating the associated processes to prevent further damage or data loss.

4 Evaluation

We conducted a series of experiments to assess the effectiveness of our proposed framework. The deception-based protection method aims to provide immediate prevention by intercepting ransomware at the precise moment it attempts to access pre-deployed decoy files.

For the evaluation of the deception-based protection method, we conducted serveral experiments. Firstly, we analyzed the proposed method's ability to accurately divide the file system, ensuring the even distribution of decoy files. This assessment was critical in determining the module's effectiveness in creating a balanced distribution.

In the second experiment, we assessed the detection performance of the deception-based module. Specifically, we examined how its detection capabilities were influenced by varying the number of partitions, which can impact the generation and distribution of decoy files.

In our experiments, a total of 94 ransomware samples, as detailed in Table 1, were utilized in the experiments. Additionally, we collected a set of 100 benign samples that performed file-related operations, including well-known compression or encryption applications like 7-zip and WinRAR. These compression applications are listed in Table 2 and were employed in the experiments, and the artificial file system used for the experiments are described in Table 3.

In addition, for the experiments, we simulated ransomware capable of conducting random-access attacks. The simulated ransomware utilizes PowerShell (Microsoft, 2023), a command-line shell, to obtain a comprehensive list of all files within the file system. Subsequently, it randomly selects a file path to access. Upon selecting a file, the ransomware proceeds to encrypt it and then deletes the original file. This cycle of file selection, encryption, and deletion continues iteratively until all files are encrypted. The ransomware was executed repeatedly, and the number of encrypted files was recorded for analysis.

Table 1: Ransomwares Samples

Name	The number of samples
Locky	14
WannaCry	49
Crypto Wall	10
Goldeneye	9
Cerber	7
Maktublocker	5

Table 2: Compression/Encryption Applications

Name	Version
7-zip	19.00
WinZip	25
WinRAR	6.02
Crococrypt	1.6
AxCrypt	2.11.6

Table 3: Artificial File System Information

Operating system	Windows 7
Root directory	C:\\
# of directories	689
# of files (text files)	9,713
Size of file system	11.7 GB

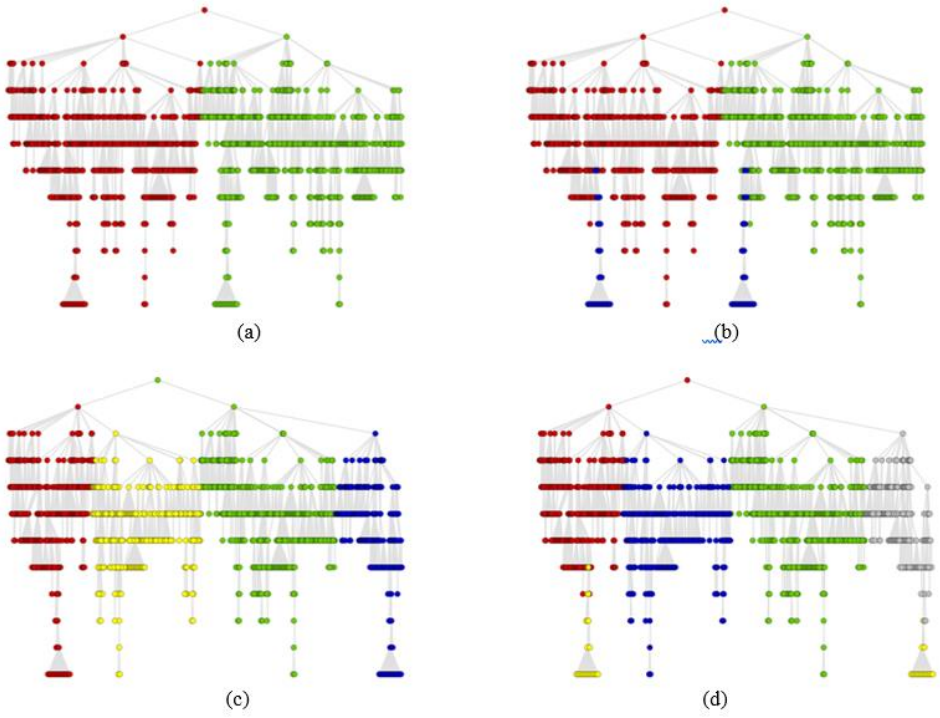

Figure 2: Sub-file Systems Partied by the Spectral Clustering Algorithm (a): 2-Cluster Partition, (b): 3-Cluster Partition, (c): 4-Cluster Partition, (d): 5-Cluster Partition (The Nodes in the Same Color Mean that they are in the Same Cluster Partition)

Performance of File System Partitioning

The proposed deception-based protection method employs clustering to determine both the number of decoy files required and the precise location for each decoy file. Consequently, we initially scrutinized the results of file system clustering to assess the method's performance accurately. The file system clustering results, as depicted in the graphs shown in Figure 2, were achieved by partitioning the file system into several subtrees using the spectral clustering algorithm. Within these graphs, each node signifies a directory within the artificial file system, and the color assigned to each node indicates its cluster type. For instance, in Figure 2(a), the red and green nodes belong to different clusters. The spectral clustering algorithm employs a similarity measure method based on the hopping distance between directory nodes. As a result, clusters are composed of directories that are near each other. In Figure 2, the subfile systems are characterized by nodes that are geographically close.

The artificial file system employed in our experiments consisted of two identical primary subtrees. Initially, a directory was created containing a total of 19,426 files and 1,378 subdirectories. This directory was then duplicated to generate another directory containing the exact same files and subdirectories as the original. These two identical directories were subsequently placed in the root directory. In this controlled environment, we anticipated that accurate file system clustering would result in the identification of two subfile systems that were either identical or very similar.

As depicted in Figure 2, our clustering results confirm the presence of the same subfile system clusters across all clustering cases. Specifically, in the instances of two and three cluster partitions, we observed one pair of identical subfile systems. In cases involving four and five cluster partitions, two pairs of identical subfile systems were identified.

Ransomware Detection Performance

The performance of the deception-based protection method was assessed using both ransomware samples and benign samples, as listed in Tables 2 and 3. This method employs hidden files as decoys, ensuring that benign programs do not interact with them. As expected, there were no instances of missed detection; in other words, the proposed method did not incorrectly flag any benign program as ransomware. Conversely, all the ransomware samples fell into the trap and approached the first decoy file located in the root directory of the artificial file system. These experimental results shows that the ransomware typically traverses directories in a depth-first search (DFS) order. In this experiment, determining whether a process exhibited ransomware-like behavior or not was a straightforward matter of implementation. While it's true that the current samples can be detected with just one decoy file in the root directory, it's essential to acknowledge that future ransomware may not consistently access the root directory first. The possibility that ransomware might employ random access patterns must be considered. To address this potential scenario, the methods for generating and deploying decoy files were specifically designed to handle assumed random-access attacks.

To provide a more precise evaluation of the proposed method's performance, we conducted an additional experiment involving newly developed ransomware that randomly accesses directories within the file system. The experimental results depicted in Figure 3 illustrate how the number of encrypted files varies as the number of partitions changes. As shown in the figure, there is a noticeable decrease in the number of files encrypted by the ransomware as the number of partitions increases. In essence, finer partitioning of the file system leads to a significant reduction in the number of corrupted files. For instance, when 100 partitions were utilized for deploying the decoy, the number of encrypted files dropped from 3,493 to 23. Furthermore, the file system could be efficiently protected using only a small set of decoy files. For example, with 18 decoy files for three subfile systems, approximately 95% of the files were protected on average. When 120 decoy files were employed for 20 partitions, around 99% of the files were protected. Table 4 provides the exact values of the number of encrypted files, which represent the average results from 100 repeated measurements. Ultimately, the adjustment of the decoy file size depends on the user's discretion. However, it is more favorable to enhance user file protection by dividing the file system into numerous partitions and evenly distributing a greater number of decoy files.

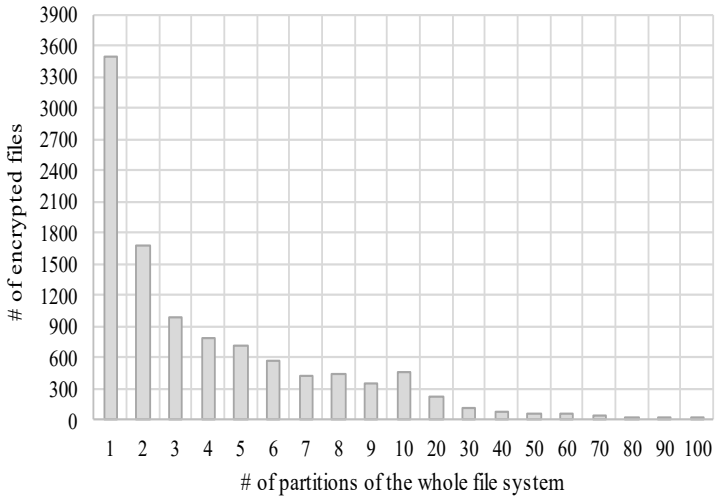

Figure 3: The Number of Encrypted Files by the no. of Sub-file System

Table 4. Experimental Results for the Random-access Attack

# of partitions	# of decoys	Avg size of a partition	# of encrypted
1	6	20,832	3493(17.98%)
2	12	10,416	1678(8.64%)
3	18	6,944	991(5.1%)
4	24	5,208	781(4%))
5	30	4,166	704(3.62%)
6	34	3,472	562(2.89%)
7	40	2,976	425(2.19%)
8	46	2,604	444(2.29)
9	52	2,314	345(1.78%)
10	58	2,083	459(2.36%)
20	119	1,041	214(1.10%)
30	172	694	107(0.55%)
40	219	520	76(0.39%)
50	276	416	62(0.32%)
60	332	347	51(0.26)
70	389	297	30(0.15%)
80	438	260	16(0.08%)
90	477	231	25(0.13%)
100	528	208	23(0.12%)

5 Conclusion

Many previous deception-based methods have faced challenges when it comes to avoiding detection through random-access attacks. This is often due to the placement of decoy files, which are typically concentrated in the root directory or in close proximity to each other. In our paper, we propose methods to address this limitation. We designed a novel deception-based approach that evenly distributes decoy files using the spectral clustering algorithm. Our experiments have shown that, on average, approximately 95% of files were protected using just 18 decoy files for three subfile systems. Furthermore, when 120 decoy files were deployed across 20 subfile systems, 99% of files were safeguarded. Our deception-based detection method is capable of providing a high level of security while requiring only a small number of decoy files, making it an efficient and effective solution.

References

[1] Al-Asli, M., & Ghaleb, T.A. (2019). Review of signature-based techniques in antivirus products. *In International Conference on Computer and Information Sciences (ICCIS)*, 1-6.

[2] Almashhadani, A.O., Kaiiali, M., Sezer, S., & O'Kane, P. (2019). A multi-classifier network-based crypto ransomware detection system: A case study of locky ransomware. *IEEE Access, 7*, 47053-47067.

[3] Al-Rimy, B.A.S., Maarof, M.A., Alazab, M., Alsolami, F., Shaid, S.Z.M., Ghaleb, F.A., AL-Hadhrami, T., & Ali, A.M. (2020). A pseudo feedback-based annotated TF-IDF technique for dynamic crypto-ransomware pre-encryption boundary delineation and features extraction. *IEEE Access, 8*, 140586-140598.

[4] Bae, S.I., Lee, G.B., & Im E.G. (2020). Ransomware detection using machine learning algorithms. *Concurrency and Computation: Practice and Experience, 32(18)*.

[5] Bayer, U., Moser, A., Kruegel, C., & Kirda, E. (2006). Dynamic analysis of malicious code. *Journal in Computer Virology, 2(1)*, 67-77.

[6] Cabaj, K., Gawkowski, P., Grochowski, K., & Osojca, D. (2015). Network activity analysis of CryptoWall ransomware. *Przeglad Elektrotechniczny*, *91(11)*, 201-204.
[7] Chakkaravarthy, S.S., Sangeetha, D., Cruz, M.V., Vaidehi, V., & Raman, B. (2020). Design of Intrusion Detection Honeypot Using Social Leopard Algorithm to Detect IoT Ransomware Attacks. *IEEE Access*, *8*, 169944-169956.
[8] Chen, B., Wang, Y.L., Gong, F.Y., Wang, X.L., & Yang, C.H. (2014). A spectral clustering algorithm for automatically determining clusters number. *In World Congress on Intelligent Control and Automation*, 3723-3728.
[9] Chen, Z.G., Kang, H.S., Yin, S.N., & Kim, S.R. (2017). Automatic ransomware detection and analysis based on dynamic API calls flow graph. *In International Conference on Research in Adaptive and Convergent Systems*, 196-201.
[10] Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., & Maggi, F. (2016). ShieldFS: a self-healing, ransomware-aware filesystem. *In Annual Conference on Computer Security Applications*, 336-347.
[11] Cusack, G., Michel, O., & Keller, E. (2018). Machine learning-based detection of ransomware using SDN. *In ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 1-6.
[12] Feng, Y., Liu, C., & Liu, B. (2017). Poster: A new approach to detecting ransomware with deception. *In IEEE Symposium on Security and Privacy Workshops*.
[13] Ferreira, A.P., Gupta, C., Inácio, P.R., & Freire, M.M. (2021). Behaviour-based Malware Detection in Mobile Android Platforms Using Machine Learning Algorithms. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 12*(4), 62-88.
[14] Gazet, A. (2010). Comparative analysis of various ransomware virii. *Journal in Computer Virology*, 6(1), 77-90.
[15] Gonzalez, D., & Hayajneh, T. (2017). Detection and prevention of crypto-ransomware. *In IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, 472-478.
[16] Jung, S., & Won, Y. (2018). Ransomware detection method based on context-aware entropy analysis. *Soft Computing*, *22(20)*, 6731-6740.
[17] Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., & Kirda, E. (2016). UNVEIL: A large-scale, automated approach to detecting ransomware. *In USENIX Security Symposium*, 757-772.
[18] Kok, S., Abdullah, A., Jhanjhi, N., & Supramaniam, M. (2019). Ransomware, threat and detection techniques: A review. *International Journal of Computer Science and Network Security*, *19(2)*, 136-146.
[19] Lee, K., Lee, S.Y., & Yim, K. (2019). Machine learning-based file entropy analysis for ransomware detection in backup systems. *IEEE Access*, *7*, 110205-110215.
[20] Maniath, S., Ashok, A., Poornachandran, P., Sujadevi, V.G., AU, P.S., & Jan, S. (2017). Deep learning LSTM based ransomware detection. *In Recent Developments in Control, Automation & Power Engineering (RDCAPE)*, 442-446.
[21] Menard, S. (2002). *Applied logistic regression analysis* (No. 106).
[22] Microsoft. (2023). *Powershell introduction*, https://docs.microsoft.com/ko-kr/powershell/scripting/overview?view=powershell-7.1
[23] Mohamad, I.B., & Usman, D. (2013). Standardization and its effects on K-means clustering algorithm. *Research Journal of Applied Sciences, Engineering and Technology*, *6(17)*, 3299-3303.
[24] Moore, C. (2016). Detecting ransomware with honeypot techniques. *In Cybersecurity and Cyberforensics Conference (CCC)*, 77-81.
[25] Nieuwenhuizen, D. (2017). A behavioral-based approach to ransomware detection. *MWR Labs Whitepaper*.

[26] Pathak, P.B., & Nanded, Y.M. (2016). A dangerous trend of cybercrime: ransomware growing challenge. *In International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, *5(2)*, 371-373.
[27] Poudyal, S., Dasgupta, D., Akhtar, Z., & Gupta, K. (2019). A multi-level ransomware detection framework using natural language processing and machine learning. *In International Conference on Malicious and Unwanted Software (MALCON)*, 1-8.
[28] Poudyal, S., Subedi, K.P., & Dasgupta, D. (2018). A framework for analyzing ransomware using machine learning. *In IEEE Symposium Series on Computational Intelligence (SSCI)*, 1692-1699.
[29] Scaife, N., Carter, H., Traynor, P., & Butler, K.R. (2016). Cryptolock (and drop it): stopping ransomware attacks on user data. *In IEEE International Conference on Distributed Computing Systems (ICDCS)*, 303-312.
[30] Securelist. (2021). *Ransomware by the numbers: Reassessing the threat's global impact*, https://securelist.com/ransomware-by-the-numbers-reassessing-the-threats-global-impact/101965/
[31] Sharmeen, S., Ahmed, Y.A., Huda, S., Kocer, B.S., & Hassan, M.M. (2020). Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches. *IEEE Access*, *8*, 24522-24534.
[32] Takeuchi, Y., Sakai, K., & Fukumoto, S. (2018). Detecting ransomware using support vector machines. *In International Conference on Parallel Processing Companion*, 1-6.
[33] Trend Micro. (2021). *Preventing WannaCry (WCRY) ransomware attacks using Trend Micro products*, https://success.trendmicro.com/solution/1117391-preventing-wannacry-wcry-ransomware-attacks-using-trend-micro-products.

Author Biography

TaeGuen Kim received the B.S. degree in electronics and computer engineering and the M.S. degree in computer and software from Hanyang University, South Korea, in 2011 and 2013, respectively. He also received the Ph. D degree in computer and software from Hanyang University, in 2018, and he worked at Hyundai Motor company as a senior research engineer. Currently, he is with Soonchunhyang University as an assistant professor since March 2021. His research interests include malware analysis, artificial intelligence, and automotive security.