RTrap: Trapping and Containing Ransomware With Machine Learning

Gaddisa Olani Ganfure, *Member, IEEE*, Chun-Feng Wu, *Member, IEEE*, Yuan-Hao Chang, *Fellow, IEEE*, and Wei-Kuan Shih, *Member, IEEE*

***Abstract*— With advances in social engineering tricks and other technical shortcomings, ransomware attacks have become a severe cybercrime affecting organizations of all shapes and sizes. Although the security teams are making plenty of ransomware detection tools, the ransomware incident report shows they are ineffective in detecting emerging ransomware attacks. This work presents "RTrap," a systematic framework to detect and contain ransomware efficiently and effectively via machine learning-generated deceptive files. Using a data-driven decoy file selection and generation strategy, RTrap plants deceptive decoy files across the directory to lure the ransomware to access it. RTrap also introduced a lightweight decoy watcher to monitor generated decoy files in real time. As the timing of the ransomware attack is not known to the victim in advance, and the ransomware encryption process is speedy, the proposed decoy-watcher executes an automatic/automated response after the detection promptly. The experiment shows that RTrap can detect ransomware with an average 18 file loss per 10311 legitimate user files.**

***Index Terms*— Deception-based detection, ransomware detection, affinity propagation, machine learning, adaptive decoy files.**

I. Introduction

IN RECENT years, digitizing items and services has empowered organizations to make data-driven business solutions and convey customized digital services to customers. By realizing the significance of data to the organizations, ransomware criminals are exploring sophisticated tactics to attack the victim's data. One emerging threat is ransomware, a malware that locks victim files by encrypting them and threatens to delete them unless the victim pays cryptocurrency. With the growing accessibility of tools for creating ransomware samples and the profit gain of the attack, ransomware attacks have matured over the years, adopting more sophisticated techniques.

In recent years, there has been considerable research on ransomware detection. The conventional approach to detect ransomware is the analysis of malicious executables [1], but such an approach is becoming ineffective against polymorphism and code obfuscation techniques [2]. To conquer the issue of code obfuscation utilized by criminals, the research community has progressively moved towards behavior-based analysis that focuses on monitoring the runtime behaviors, such as system call sequence and R/W operations, which are hard to modify without changing the core functionality of the malware [3], [4]. For instance, in process level monitoring, any running process with a trust score higher than the maximum threshold value will be addressed as malicious activity. However, an adversary can bypass these approaches by splitting the ransomware activity into N different processes, where each process performs a small fraction (e.g., $1/N$) of the total ransomware operations with the same net output of executing ransomware [5]; for example, one particular ransomware family that uses this evasion technique is LockerGoga [6]. The behavioral-based detections are effective with the known threats but not unseen ransomware samples. As ransomware attacks user files, recent works introduce a decoy file-based deceptive solution to detect cryptographic ransomware attacks early. These decoy files (or fake files) are created automatically or manually and planted in the entire network [7].

The current ransomware detection techniques are constantly confronted with new challenges. The main technical challenges of detecting emerging ransomware include the emergence of file-less ransomware, utilization of multithreading, file prioritization strategies, and code obfuscation for encryption. Notably, in 2019, we saw the most substantial number of ransomware incidents ever [8], wherein in the U.S. alone, more than 966 organizations were affected by ransomware attacks with an approximated cost of $7.5 billion. Shockingly, 77% of organizations impacted by ransomware were running state-of-the-art endpoint protection [9]. The report underlines that existing solutions to the organization's security posture are inadequate to cope with emerging ransomware threats. Such an observation motivates us to look for an effective and efficient strategy that provides a post-breach ransomware detection and mitigation solution agnostic to the attack's ransomware family or variant.

This work proposes a ransomware detection and containment strategy called *"RTrap,"*, based on a simple yet effective concept of *trapping and containing ransomware with machine learning*. In contrast to prior works, the decoy files in RTrap

Manuscript received 1 July 2021; revised 25 April 2022 and 12 August 2022; accepted 11 January 2023. Date of publication 26 January 2023; date of current version 7 February 2023. This work was supported in part by the National Science and Technology Council under Grant 111-2223-E-001-001, Grant 111-2923-E-002-014-MY3, Grant 111-2221-E-001-013-MY3, Grant 112-2927-I-001-508, and Grant 112-2222-EA49-002-MY2; in part by the Academia Sinica under Grant AS-IA-111-M01 and Grant AS-GCS-110-08; and in part by the Ministry of Education under Yushan Young Fellow Program. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Pete Burnap. *(Corresponding authors: Yuan-Hao Chang; Gaddisa Olani Ganfure.)*

Gaddisa Olani Ganfure is with the Department of Computer Science, Dire Dawa Institute of Technology, Dire Dawa 3000, Ethiopia (e-mail: gaddisaolex@gmail.com).

Chun-Feng Wu is with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu City 30010, Taiwan (e-mail: cfwu417@cs.nycu.edu.tw).

Yuan-Hao Chang is with the Institute of Information Science, Academia Sinica, Taipei City 115024, Taiwan (e-mail: johnson@iis.sinica.edu.tw).

Wei-Kuan Shih is with the Department of Computer Science, National Tsing Hua University, Hsinchu City 300044, Taiwan (e-mail: wshih@cs.nthu.edu.tw).

Digital Object Identifier 10.1109/TIFS.2023.3240025

are chosen from legitimate user files adaptively through a data-driven machine learning algorithm. Given N user files, RTrap adaptively picks K subsets of user files that efficiently and effectively represent the collection of documents as representative decoy files.

RTrap initially considers a directory's entire files as a potential decoy file. A good set of exemplar decoy files emerges after exchanging attraction messages through the affinity propagation algorithm. After successive iterations, K user files that receive the highest vote will become representative decoy files. Then, the subset of legitimate user files selected in this manner is duplicated and renamed to become representative decoy files. Besides, to ensure that the generated decoy files are up-to-date, RTrap periodically checks for each update to a directory and reflects user activity on each decoy file whenever necessary. The decoy-watcher monitors the decoy files in real-time and flags each potential access to the registered decoy files as suspicious activities; it can apply some remedial actions to stop the attack in less than 5 s in most cases and less than 11 s for more sophisticated ransomware families. The experiment result also shows that RTrap can achieve a 99.42% deception rate (i.e., an average of 18 file-loss out of 10311 user files) over different families of ransomware.

In short, the decoy-file selection and generation strategy of RTrap is different from other approaches in various ways. First, decoy files are selected from legitimate user files by employing the affinity propagation technique, i.e., a technique that allows the exchange of attractiveness messages among files in a directory. While experimenting using 20 ransomware families, we observed that no single ransomware sample distinguishes the decoy files we planted from the legitimate users. The existing deception strategy does not consider emerging ransomware's multithreading and file prioritization strategy. However, the experiment shows that the deception quality of RTrap is consistent across the ransomware families, even with those that employ multithreading. The proposed methodology of planting decoy files aims to maximize the chance that the ransomware modifies the decoy files at all times. To ensure the freshness of the decoy file, RTrap introduces a mechanism to continuously regenerate decoys to prevent an adversary from learning the deployed decoy files. This idea can be realized by periodically scanning the drive for a chance to update the decoy files. The existing deception-based strategy proposes a fixed number of decoy files. However, our Affinity Propagation-based strategy will avoid the need to make a static number of decoy files by intelligently learning the best number of decoy files for protection.

The rest of this paper is organized as follows. Section II presents the background and related work, and Section III talks about the observation and motivation of this work. Section IV introduces RTrap, a strategy to detect and mitigate cryptographic ransomware attacks. The overall performance of the proposed design, both in terms of deception quality (i.e., detection rate) and efficiency, is assessed and discussed in Section V. Finally, Section VI summarizes the work and layouts some future directions.

II. Background and Related Work

A. Ransomware

Ransomware is a malware class that locks victim files by encrypting them for financial gain. The encryption and social engineering tricks utilized by the ransomware samples vary; however, most of the activities utilized to infect the victims are the same across the families. First, the ransomware actors use social engineering strategies to deceive users into downloading a dropper from a compromised link to their computers. In the background, the dropper downloads an executable that installs the ransomware. Then, the ransomware scans the local and network-attached storage devices for particular file types to encrypt. Following the file shortlisting, the ransomware prioritizes and encrypts all the files. Finally, a readme file containing the payment information will be shown to the victim. The entire process happens rapidly, and more advanced ransomware families can also spread to network-attached devices. Note that the costs associated with a ransomware attack are not only the direct cost of paying the ransomware criminals but also the cost of enforced downtime, reputation loss, and liability.

Besides the victim mistake, some technical traits make it harder to detect ransomware attacks:

1) File search and encryption processes employed by ransomware are normal file-related operations.
2) Ransomware uses a secure encryption algorithm that is impossible (or difficult) to break.
3) Accessibility tools to make ransomware variants allows criminals to use zero-day exploitation (or new variant) attacks.
4) By attaching its malicious activity to the legitimate process, ransomware can bypass endpoint protection.
5) The ransomware attacks operate quickly since they target specific file types for encryption.

Overall, the net of these five points indicates that detecting ransomware attacks is challenging.

Although the challenge and traits of ransomware differ from family to family, they still interact with the user files. Hence, by promptly detecting the interaction between ransomware, and victim files, it is possible to overcome the issue of *late detection* and *code obfuscation* strategies introduced by emerging ransomware.

B. Related Works

In the literature, plenty of ransomware detection methods have been proposed and each technique falls into one of three categories: a) *method based on ransomware signature*, b) *method based on real-time behavior analysis*, and c) *deception-based detection*.

Signature-based detection is a widely used technique to detect malware/ransomware. This approach identifies opcodes or strings that match the known ransomware marks. Authors in [1] propose a static analysis framework that uses N-gram opcodes with deep learning for ransomware detection. They treat N-gram analysis of sequences as a natural language processing problem, and their finding shows that their model

is good at classifying ransomware activities. Moreover, the authors in [10] proposed a ransomware detection strategy that utilizes reverse engineering and static analysis of binary codes with machine learning. Their finding shows that their model exactness is nearly 96.5% on average. The big problem with signature-based detection is that it fails to recognize obfuscated code or ransomware that does not have signatures. Technically, adding a single byte to the executable file will create a new hash, allowing ransomware to bypass signature-based detection.

Most ransomware detection solutions rely on dynamic behavioral analysis of executables such as file-system interaction patterns, traffic analyses, API calls, registry activity, process activity, and hardware events (i.e., hardware performance counters). Authors in [11] introduced pre-attack Paranoia Activities such as API calls for ransomware family attribution. To fingerprint the API activities, they execute the ransomware collected from VirusTotal and VirusShare inside the Cuckoo sandbox and then apply word-level natural language processing (NLP) algorithms to represent and extract API features. Following the feature extraction, they investigated numerous machine learning models such as Naive Bayes, k-NN, RF, NN, LSTM, and Bi-LSTM. Their finding shows that a model built using Random Forest achieved an average classification accuracy of 95%. A work in CryptoDrop [12] utilized the file system activities such as I/O request patterns and file entropy to detect ransomware attacks.

The interaction between C&C Server and the victim can likewise be used as a behavioral feature to detect ransomware attacks [13], [14], [15], [16]. For instance, Almashhadani et al. [15] investigated Locky Ransomware using network traffic data. The authors extracted 20 features from the network traffic data to build a classification model. Their experimental evaluation of the proposed detection system using Random Forests, Bayes Networks, and Support Vector Machines demonstrates that it offers high detection accuracy, low false positive rate, useful extracted features, and is highly effective in tracking ransomware network's activities. Since their work only focuses on locky ransomware, it is not easy to generalize the finding as it may not work for others. Also, while some ransomware families require an Internet connection to start the encryption process, most ransomware families need not require the C&C server connection to start the encryption process on the victim file, which makes this approach relatively constrained.

Another behavior-based ransomware detection feature is monitoring hardware performance counters (HPCs) such as cache-reference, cache misses, and several instructions [17], [18]. To gain more ransom, ransomware criminals try to encrypt as many user files as possible. Aggressively performing file encryption and renaming usually incurs context switch and thus fluctuates CPU status, such as CPU cache and branch prediction. For instance, using several machine learning algorithms, authors in EGB [18] evaluate the features obtained from hardware performance counters to classify malicious applications into ransomware and non-ransomware categories. Their finding depicts that the model built using Random Forest achieved the highest detection rate.

DeepGuard's [19] work proposes a mechanism to capture user activity by monitoring file interaction patterns and uses a deep generative autoencoder architecture to recreate the input. Their primary assumption is to train the deep generative autoencoder with the dataset of user file-interaction log collected for several days. With repetitive training, the model will be skillful in reconstructing unseen inputs with minimal reconstruction error when the provided information resembles legitimate user activity or patterns used to train the model, and hence it can spot anomalous (or ransomware) activity. In short, by solely modeling the user interaction pattern, this approach can discriminate ransomware activity from benign one.

In [20], a ransomware protection mechanism called SSD-insider++ was proposed. This system worked in two folds, i.e., detection and data recovery. This study was limited in its implementation as assessing every I/O block, its header, and payload during run-time is infeasible. The detection algorithm observes I/O patterns of a host system and decides whether the host is being attacked by ransomware in an early stage.

In order to improve the detection rate of crypto-ransomware, authors in File-entropy [21] used the encrypted file entropy to classify spot ransomware activity. More than 20 file formats are encrypted and analyzed by WannaCry, Phobos, GandCrab, and GlobeImposter ransomware to gather features essential to differentiate the ransomware task from that of file encryption, such as Zip and 7z. Those extracted features are provided to the support vector machine for the classification tasks. The experiment result shows that their model can detect ransomware activity with 85.17% average accuracy. However, since this work only accesses only four ransomware samples, it may not be capable of detecting other classes of ransomware.

The deception-based solution is the third approach to detecting ransomware attacks. The deception-based solution works by planting fake computer system assets (e.g., a web server or router across the enterprise) to let the intruder come across and trigger an alert whenever those deceptive assets are accessed. Despite being in its infancy, some recent work proposed a deception-based strategy to detect ransomware attacks [22], [23], [24], [25]. To enable decoy-based ransomware detection, Lee et al. [22] introduced a proof of concept to create and place a decoy file based on ransomware file traversal pattern. In their experiments, different ransomware samples are decompiled and analyzed to understand ransomware file system traverse patterns. Based on their analysis, they suggest planting two deceptive files in the root directory (one decoy file, which comes first in alphabetical order, and another decoy file that comes first in reverse order). Besides, Gómez et al. [23] developed a tool named R-Locker to impede the presence of ransomware attacks. In their work, decoy files are generated as a named pipe using *makeinfo* command in the Unix system. These files are overwritten with 3 KB data and placed in the user home directory, and the R-Locker blocks any process that attempts to access those files. However, as all the generated

decoy files are of the same type, ransomware can avoid encrypting those named pipes so that the purpose of deception-based detection is nullified. In Rwguard [25], the authors unified the behavioral-based detection model with the decoy-based approach to mitigate cryptographic ransomware attacks. In Rwguard, the original user files are randomly chosen as a decoy file. Furthermore, Cryptostopper [24] places arbitrarily generated decoy files in the file system to provide a deceptive solution against ransomware attacks. Hence, whenever any malicious process tries to change the content of deceptive files, Cryptostopper creates an alert to notify the system administrator and close the infected host. The Cryptostopper is a commercial product, so the technical details were not reported.

In general, existing deception-based ransomware detection approaches assume that the ransomware uses a traversal sequential file system to encrypt the file. However, our observation indicates that emerging ransomware families (e.g., Ryuk) utilize multithreading and file-prioritization strategies to encrypt victim files in parallel, leaving existing solutions ineffective in detecting emerging ransomware attacks.

III. Observation and Motivation

A. Observation

The contention between ransomware criminals and security teams is continuously heating up as both create new weapons in their arsenal. As the security experts develop a new ransomware detection strategy, ransomware actors develop a more sophisticated technique to bypass the detection system. In recent years, there has been considerable research on ransomware detection. However, the current ransomware detection techniques are constantly confronted with new challenges. The typical emerging challenges include (1) code obfuscation, (2) file-less ransomware, (3) multithreading, and (4) file prioritization for encryption.

1) Code Obfuscation: The utilization of sophisticated *code obfuscation and polymorphism* are sharply increasing [26]. This technique allows criminals to hide the well-known string of malicious code to bypass the detection system. Thus, it is less likely to detect obfuscated code using signature-based detection. Furthermore, during the dynamic analysis, the signature-based solutions perform memory-intensive tasks such as intercepting and correlating the API call sequence with malware signatures or tracking each process activity for digital evidence; this approach is computationally expensive. Most organizations impacted by ransomware were also running state-of-the-art signature-based and behavioral-based endpoint protection [9].

2) File-Less Ransomware: In addition to the code obfuscation techniques utilized to evade the detection [26], the shift to *fileless ransomware* is also increasing in the cyber community. Unlike attacks with files, *fileless ransomware* is sneakier in its malicious activity into legitimate applications already built into the operating system so that it turns Windows against itself. For instance, ransomware families such as MegaCortex use stealth to abuse window system components such as PowerShell to start the encryption automatically to make the attack appear to come out of nowhere [27]. In this approach, the trusted Windows process performs a file encryption task, making endpoint protection software believe that a trusted application is modifying the documents. Overall, file-less ransomware poses two main challenges to existing antiviral solutions, (1) without an executable, there is no footprint for signature-based solutions to detect this type of attack, and (2) file-less ransomware works in memory. As a result, its activity terminates when the system closes, making it more challenging for security personnel to uncover what happened; however, uncovering what happened is essential to prevent future attacks. Thus, even organizations are deploying active firewalls and applications allowing detective solutions; using this stealth strategy will evade the security perimeter and leave existing solutions ineffective [28].

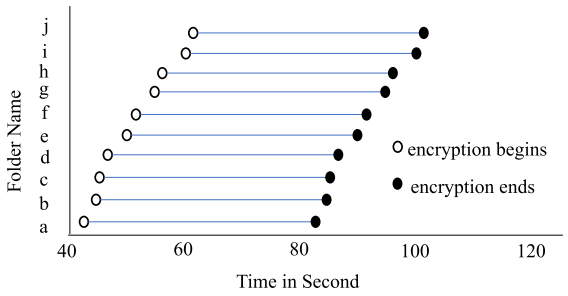
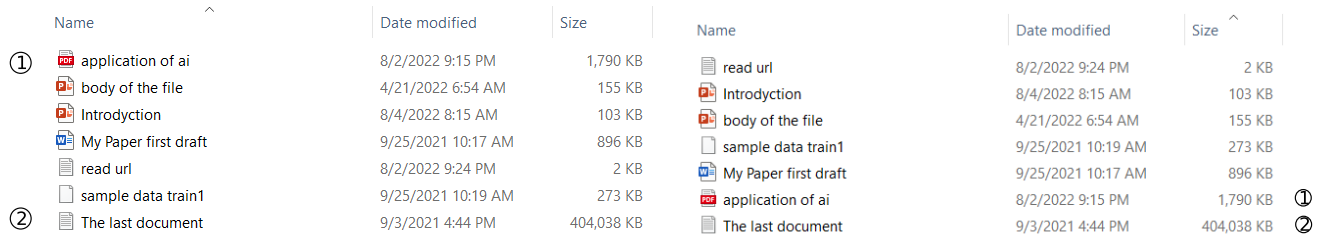


Fig. 1. Multithreading Property of Emerging Ransomware (e.g., Ryuk).

3) Multithreading: Existing decoy file-based deception strategies are built on the view that a ransomware attack launches a single process so that the encryption is done linearly from a root directory to subdirectories. Taking into account that the ransomware file traversal begins from the root directory, they recommend placing a decoy file in a root directory. However, modern computers now have more than one multi-core processor with *multithreading* technology. Such advances in chip design offer tremendous performance benefits for everyday business activities, as they allow parallel execution of a process to accelerate productivity. On the criminal side, recent ransomware is designed to efficiently use modern CPU hardware (if present) to parallelize the task of directory traversal and file encryption, subsequently guaranteeing faster and more destructive impact before being detected by victims [29]. For instance, a class of ransomware families such as Ryuk and LockerGoga utilizes a *multithreading* strategy to launch a sub-process for each shortlisted directory to fasten the encryption process and make it difficult to detect by existing solutions [29]. To validate this, we execute one of the Ryuk samples in a virtual machine and log its activity on a directory containing ten folders, each with 98 diverse document types. The output in Figure 1 results from multiple threads, each of which handles a different directory to encrypt user files in parallel, for example, from 60*s* to 80*s*. This strategy enables the ransomware to bypass process-level monitoring and central decoy file placement strategies. This attack can achieve higher throughput and lower latency for the attacker; thus, a more robust design is required.



(a) Defender Create decoy files Based on File Name. (b) Attacker Sort Files Based on File Size.

Fig. 2. Observation: decoy file Creation Strategy vs Ransomware Prioritization Strategy.

4) File Prioritization: The other key features of emerging ransomware families are directory and file *prioritization strategy* (or *data-centric property*). As the documents in the enterprise can be stored in fixed local storage, removable drives, or mapped remote shared drives, the ransomware might prioritize certain drives or document sizes first to guarantee success before being detected by endpoint protection systems. In addition to storage drive prioritization, the ransomware can be programmed to prioritize documents based on file metadata such as size, date, and type. Example ransomware that employs prioritization is BlackRuby, which first enumerates all files and encrypts those denylisted file types that are hardcoded in the binary and have a file size less than 512 MB [30].

However, existing decoy file generation strategies do not consider this *data-centric* behavior of ransomware. For instance, in [22], authors suggest creating two decoy files: one comes first and the other comes at the end after file name based sorting to overcome traversal techniques utilized by ransomware. To show the inadequacies of this strategy, we persuade it by showing one example directory content that appeared in Figure 2. The directory content shown in Figure 2(a) consists of 20 legitimate user files and two decoy files created according to the filename. In a scenario where the ransomware employs the same sorting strategy as the way the decoy files are created (in this example, filename), the deception guarantees 100% protection. However, as the ransomware file-ordering/prioritization differs for each family or variant, *relying solely on specific file attributes is not effective in detecting a variant class of ransomware.* As appeared in Figure 2(b), if the ransomware sorts the directory content shown in Figure 2(a) based on file-size in ascending order, the two decoy files shown in the directory will be encrypted toward the end which nullifies the purpose of the deception. These all indicate that the emerging ransomware utilize different tactics to prioritize the encryption process, and thus *creating a decoy file statically by considering only specific file attribute will not suffice as it leads to late discovery, subsequently, more file-loss and more ransom demand for restoration*.

B. Motivation

Prevention is an ideal solution against a ransomware attack. However, due to the victim's mistake and technical challenges, most of the attacks cannot be prevented. For instance, as the report indicates, nearly half-million malicious binaries are discharged every day, and notably, 99% of them have no known signatures (i.e., zero-day attack) [31] making prevention obsolete for most attacks. When the attacker circumvents the prevention strategies, the next line of protection is the early detection and containment of the attack. By early detection, the defender can hinder the attack, or take other defensive actions. Numerous solution has been proposed in the literature to detect a ransomware attack. However, the report indicates that the majority of organizations impacted by ransomware were also running state-of-the-art endpoint protection software [9], which underlines that existing solutions to detect ransomware attacks are not adequate to cope with emerging ransomware threats.

Based on the emerging traits of ransomware attacks observed in Section III-A, this work is strongly motivated by the need to develop effective and efficient strategies to combat the various strategies used by ransomware. We first conduct a ransomware file-interaction analysis to look for an alternative strategy that is resilient to ransomware variants. By executing the ransomware families inside the sandbox and logging their activity, we discovered some common ransomware traits that can be useful for detecting all types of attacks. That is, regardless of the way ransomware utilizes code obfuscation or stealth strategy to bypass security perimeter, ransomware can change and rename a list of files (i.e., encryption and renaming of the files with specific file extension), create or remove a list of files (i.e., copying the original file, encrypting it, and finally deleting the original content). This point out that, although the challenge and traits of ransomware differ from family to family, they still interact with the user files. Encrypting the entire victim files will take time to complete. Hence, by promptly detecting the interaction between ransomware and victim files, it is possible to overcome the issue of late detection and code obfuscation strategies introduced by emerging ransomware.

IV. RTrap: Ransomware Detection and Containment Strategy With Machine Learning

A. Overview and Design Concept

In this section, we introduce a design concept of "RTrap," a systematic way to detect and contain ransomware by luring ransomware to attack the adaptively generated decoy files (or traps). In essence, our method considers the variation in file sorting strategy and the multithreading property of emerging

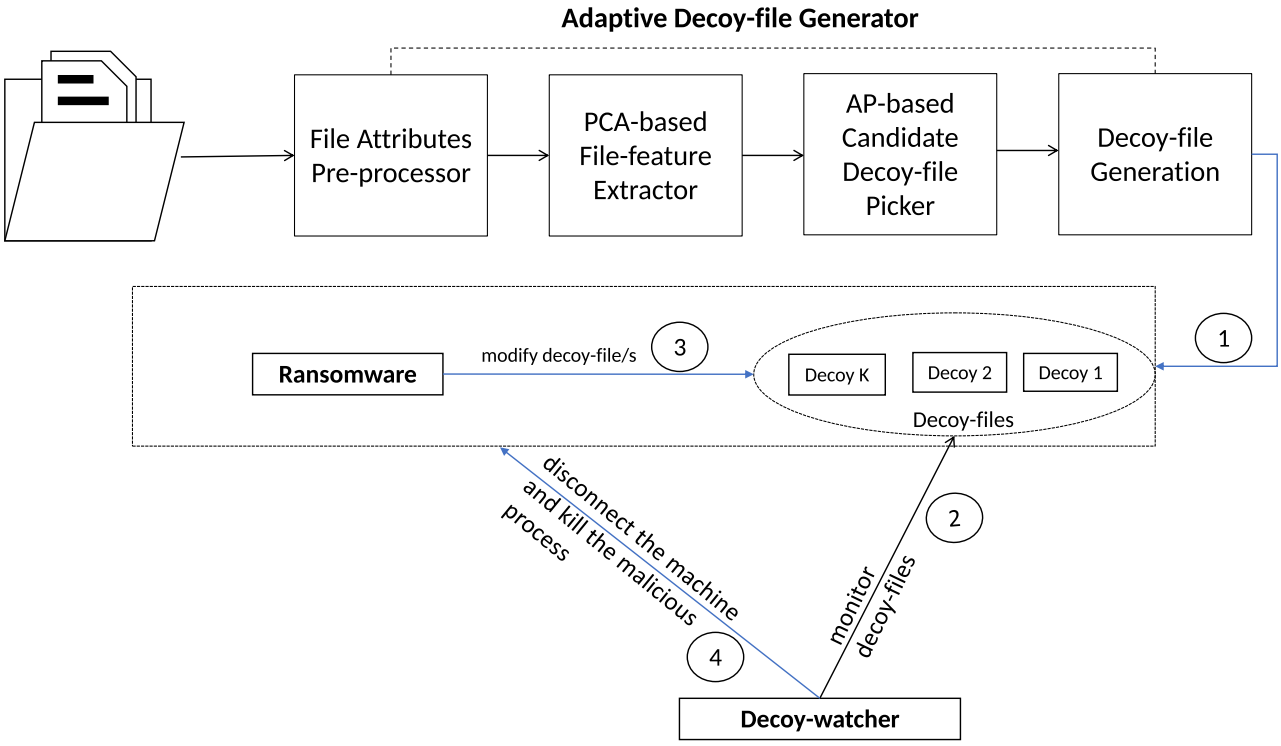


Fig. 3. Overview of RTrap.

ransomware to generate deceptive files at the endpoint (i.e., our solution provides endpoint security). We achieve this by scanning the entire endpoint directory iteratively and allowing all files in each directory to vote for candidate decoy files based on their interestingness measure. After the successive exchange of attractiveness messages (i.e., file attributes such as file type, file size, date created, and date modified), our strategy selects subsets of user files as decoy files. The adaptive nature of decoy file selection based on file attributes of the directory and the utilization of legitimate user files as a decoy file makes the proposed solution have good performance across ransomware families. Also, utilizing legitimate user files as a decoy file for each endpoint makes it difficult for adversaries to distinguish the decoy files from the legitimate ones. RTrap ensures the freshness of the planted decoy files by period scanning the respective directory for change. In RTrap, users are believed to be aware of the decoy files on their machines. Thus, any modification actions to the decoy files signal a ransomware presence. The decoy watcher components of RTrap are responsible for monitoring the planted decoy files. Upon detecting modifications to the deployed decoy files, the decoy-watcher will automatically disconnect the host machine from the network and kill the malicious process.

As shown in Figure 3, "RTrap" includes two major components, i.e., *Adaptive Decoy-file Generator* (see Section IV-B) and *Decoy-watcher* (see Section IV-C). Given N files of a directory, RTrap (referred to as the "RTrap model") adaptively selects the K subset of user files that efficiently represent the collection of documents as representative decoy-files by employing a data-driven machine learning algorithm, i.e., Affinity Propagation (AP). Unlike the other clustering algorithms (e.g., K-means), the AP algorithm does not require a prior estimation of the value of K [32]. Hence, it best suits for *"RTrap* to intelligently decide the required number of decoy-files. Once the decoy files are generated, the decoy-watcher component monitors them in real-time so that any potential access to them will trigger defensive actions to contain the attack (i.e., disconnecting the victim from the network and killing the malicious process immediately).

Adaptive selection and real-time monitoring of decoy-files will empower the proposed model to detect early the potential attempts to access the decoy-files (including *multithreaded* execution). Moreover, as the model selects decoy files by considering all the file attributes, the impact of the file prioritization strategy utilized by emerging ransomware (i.e., *data-centric property*) is minimal. Besides, to ensure that the generated decoy files are *up-to-date*, the model periodically checks for any update to a directory and reflects user activity on each decoy file whenever necessary. Overall, irrespective of the ransomware variant performing the attack, the proposed solution can detect and contain the malicious process performing the attack as far as interacting with the decoy files.

B. Adaptive Decoy-File Generator

The decoy file generator module scans the target machine's file system starting at the specified root and identifies all folders containing more than three files (i.e., the number can

be adjusted), then select and plant decoy files for the respective folder.

As illustrated in Figure 3, the decoy file generation module comprises of File-attributes Pre-processor (see Section IV-B.1), Principal Component Analysis(PCA) based File-feature Extractor (see Section IV-B.2), AP-based Candidate Decoy-file Picker (see Section IV-B.3), and Decoy-file Generator (see Section IV-B.4). First, the *"File-attribute Pre-processor"* component in "RTrap lists all files in a directory and represents each file as a vector of attributes (e.g., file size in Byte). As the type of value that each file-attribute store differs, they are standardized to reduce the influence of a high-valued attribute on the subsequent decoy file selection algorithm. Then, the "PCA-based File-feature Extractor" module then projects the standardized file attributes into a new representation by linearly combining the original unsupervised inputs. The new representation outputs are uncorrelated and ranked based on their significance (i.e., explained variance). Therefore, by constraining the cumulative explained variance, this component can expel redundant features while preserving important features. The dimension-reduced data becomes the input to the "AP-based Candidate Decoy-file Picker," which applies a clustering algorithm to return a list of K candidate decoy files. Finally, based on the output of the AP-based Candidate Decoy file Picker and predefined constraints (e.g., the maximum file size limit of decoy files), the selected legitimate user files are renamed and created for each directory by the "Decoy-file Generator."

Note that deception-based detection has a narrow field of view; when the ransomware file prioritization strategy is not similar to that of the deployed decoy-file attributes, the decoy-watcher may never detect the ransomware, or it may detect it lately after several files are encrypted. Thus, planting decoy-files heuristically with specific file attributes as in [22] will constrain the deception performance (or quality). However, the data-driven decoy-file selection strategy of "RTrap allows it to introduce variety in both the number and types of files (or file-attributes) to use for each directory (see Figure 5). Subsequently, it generates the best decoy files that have the potential to be accessed at an earlier stage compared to the heuristic selection of decoy files. Thus, we consider that the deception quality of the proposed solution is resilient to the ransomware file-prioritization strategy.

1) File-Attributes Pre-Processor: The decoy file generation process begins with transforming all files into a vector of features in a directory. In this work, a feature is a file attribute (specifically, file size, file type, file creation date, and date modified) that can be extracted from a file using File-attributes Pre-processor. By naively looking at those features, one can observe that the type of value that each attribute store differs; for example, the file size stores numeric value, whereas the file type stores a category. Thus, it is essential to standardize those attributes to some scale so that the value of one attribute does not dominate the decoy file selection process.

A data pre-processing is applied to normalize the attributes to make them suitable for the decoy file selection component. In "RTrap, continuous-valued attributes (e.g., file size) are normalized using standard-scaler with mean zero and unit variance; categorical attributes (e.g., file type) are integer encoded; date and time-related file attributes (e.g., modification date) are standardized using sine and cosine transformation [33], [34]. Subsequently, for each file in a directory, the following standardized features are formed [file-type, file-size, date-created, $\ldots$, f_m], where m indicates the number of attributes or features after standardization. Note that the ransomware utilizes different prioritization techniques to shortlist the victim files for encryption (see Section III). Note that it is essential to incorporate the most common file attributes because the file sorting attribute that the cybercriminals use is not known a priori. Finally, a feature matrix of the form $N \times M$ will be formed for each directory, where N indicates the total count of user files in a directory, while M represents the count of file attributes (or features) after standardization. Note that after performing the file-attributes pre-processor, we refer to "file" as "*data point*," and a collection of data points is denoted as a feature matrix D which corresponds to one row in D_f. The set of all features in a directory D is denoted as D_f.

2) PCA-Based File-Feature Extractor: To have helpful machine learning models (or results), we must find a simple yet effective way to represent the input feature while preserving the representation and reducing the computation overhead. This concept is also critical for the proposed decoy file picker module as it employs a distance metric to calculate the initial similarity matrix. Intuitively, as the size of feature matrix D increases, the computational overhead increase; this also adversely affects the clustering result [35] during the decoy file selection phase. For instance, some directories may contain the same file type or creation time. In such cases, including repetitive attributes will pose additional computational complexity with no gain in clustering performance. Thus, our design expels those redundant features by applying a PCA-based File-feature Extractor. This module applies a principal component analysis[1]("PCA" for short) to project the standardized feature matrix into a new representation that is formed by linearly combining the original inputs in an unsupervised way. Given a normalized feature matrix D, the PCA-based File-feature Extractor works: first, calculate the covariance matrix between a pair of data points. Then, decompose the covariance matrix into *eigenvalue* and *eigenvectors*, where the eigenvector indicates the direction of transformation (or line), whereas the eigenvalue indicates the spread of data points on the line which is an eigenvector. Finally, the new input representation (D_{new}) is calculated as follows:

$$D_{new} = W^T D \tag{1}$$

where D is the original data matrix and W is a matrix of principal component (or eigenvectors). Note that unless some threshold is set a priori, Equation 1 returns the same number of features as in the original data matrix but sorted in their importance value from left to right. Thus, the remaining task is to pick L representative principal components (where $L<N$) represent the data well (i.e., which have the most variance).

[1]PCA is a de facto algorithm to reduce input dimensions in various machine learning applications such as face recognition [36], image denoising [37]

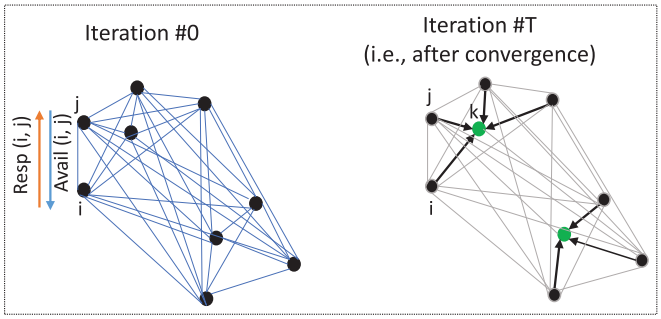


Fig. 4. Example Message Graph of decoy file Selection Process (The color of the arrow after iteration T indicates the strength of the message for a point i to select a candidate point k as a representative point (or decoy file)).

In this work, the number of principal components is selected online based on their explained variance, which is the proportion between the variance of each principal component and the total variance. In the prototype of "RTrap, the variance threshold is set to 99%, which indicates that the principal component (or feature) whose values do not change from observation to observation are expelled from the subsequent task. This strategy is essential to reduce unnecessary features with an explained variance near zero, such as having the same file attribute for all files in a directory. Overall, by enforcing the threshold on the output of PCA, the PCA-based File-feature Extractor module remove redundant features (or attributes) and thus can effectively extract the discriminant feature, which is essential for the subsequent clustering algorithm. Thus, the final output of this component is a new feature matrix with a reduced dimension of $N \times L$, where L denotes the chosen number of principal components.

3) AP-Based Candidate Decoy-File Picker: Suppose that there are N files in a directory, and each file in a directory is of the form x_i, $i = 1, 2, \ldots, l$ (where l is the number of features after the PCA-based file-feature extraction). The design goal of the decoy file picker component is to identify clusters of similar data points (i.e., clusters of similar files) based on the extracted features and return K data points that best represent the clusters; each returned K data point is a cluster center that later becomes a candidate decoy file. However, there is one basic constraint in the unsupervised clustering algorithm, and this constraint is that the desired number of groups (or clusters) should be specified (e.g., K, where $K < N$). Choosing K samples out of N possible samples is a combinatorial optimization problem that is NP-hard [38]. As a means to tackle this challenge, our solution adopts the algorithm called Affinity Propagation (AP) [32].

AP-based center selection works by simultaneously considering each data point as a potential center and allowing each data point to vote for its representative data by employing similarity metrics. Finally, a subset of data that receives a vote will be considered a center. Thus, the value of K is equivalent to the number of data points receiving the vote (or receiving a preference from other data points).

Likewise, the AP algorithm solves the candidate decoy-file selection problem by initially considering each data point as the potential decoy-file. Then, after successive message passing between candidate decoy-files, a good set of decoy-file emerges. Hence, in our design, the candidate decoy files (i.e., exemplars) and the number of decoy files (i.e., K) are intelligently decided by the model. This idea can be conveyed mathematically by first calculating the similarity ($Sim(i, j)$) between a data point x_i and another data point x_j with a convectional negative Euclidean distance metric as follows:

$$Sim(i, j) = -\|X_i - X_j\|^2, i \neq j \tag{2}$$

Upon computing the pairwise similarity, the next step is enabling the exchange of two types of attractiveness messages among the feature vectors until the desired convergence level is reached (see Figure 4). The first message transmits from point i to the candidate point j is the responsibility matrix $Resp(i, j)$, which measures how appropriate a point j can represent a point i, considering other competing points. The subsequent message is the availability matrix $Avail(i, j)$, which estimates how suitable it could be for a point i to pick point j as its representative center, considering the support that data point j receives from other potential data points. In the beginning, the value of both the responsibility and availability matrix is initialized to zero. Then, for each iteration, the corresponding value of both matrix is computed as follows [32]:

$$Resp(i, j) = \begin{cases} Sim(i, j) - \max_{j' \neq j}\{Avail(i, j') + Sim(i, j')\}, & i \neq j \\ Sim(i, j) - \max_{j' \neq j}\{Sim(i, j')\}, & i = j \end{cases} \tag{3}$$

$$Avail(i, j) = \begin{cases} \min\{0, Resp(j, j)\} + \sum_{j' \neq i, j} \max\{0, Resp(j', j)\}, & i \neq j \\ \sum_{j' \neq i} \max\{0, Resp(j', j)\}, & i = j \end{cases} \tag{4}$$

As given in Equation 3, $Resp(i, j)$ quantifies the relative similarity between data point i and j considering how similar other data point j' is to i ($Sim(i, j')$) and how available j' is to i ($Avail(i, j')$). Thus, the value of $Resp(i, j)$ only increases whenever the availability of other competing j' decreases. On the other hand (Equation 4), the availability of j to i ($Avail(i, j)$) is the function of how responsible a data point j is to itself ($Resp(j, j)$) and how positively responsible j is to the other data points ($\sum_{j' \neq i, j} \max\{0, Resp(j', j)\}$).

Moreover, to ensure model convergence, the computed values after each iteration are updated as follows:

$$Resp(i, j) = (1 - \alpha) Resp(i, j) + \alpha Resp_{t_0}(i, j) \tag{5}$$

$$Avail(i, j) = (1 - \alpha) Avail(i, j) + \alpha Avail_{t_0}(i, j) \tag{6}$$

where $Resp_{t_0}(i, j)$ and $Avail_{t_0}(i, j)$ represent previous values of responsibility and availability, whereas α is a learning rate introduced to reduce numerical oscillations. Finally, the model returns the best exemplar/representative point (or candidate decoy file) for each point i, so that the sum of availability and

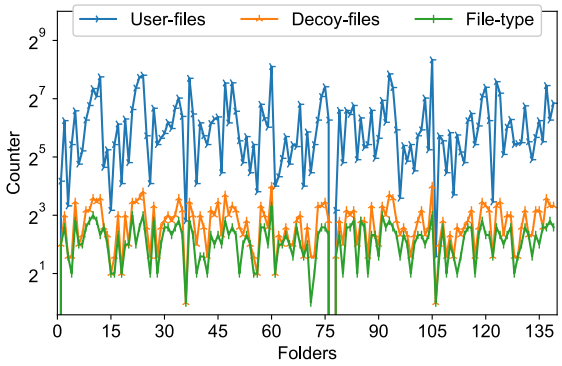


Fig. 5. Example to Show the Adaptivity of decoy file Selection Strategy (Variations in number and types of decoy files selected across the folders).

responsibility is maximized with the following equation:

$$Exemplar_i \leftarrow \arg\max_k \{Resp(i, j) + Avail(i, j)\} \quad (7)$$

Upon convergence, the model returns k exemplars (or candidate decoy files) with their index and passes them to the decoy file generation module.

As shown in Figure 5, the model dynamically selects a varying number of decoy files, in terms of number and file-types across the directories. Since the proposed design is data-driven, the variety appeared in Figure 5 is because of the variety in file types and the number of legitimate user files across the directory.

As shown in Figure 5, the model dynamically selects various decoy files in terms of number and file types across the directories. Since the proposed design is data-driven, the variety that appeared in Figure 5 is because of the variety in file types and the number of legitimate user files across the directory.

4) Decoy-File Generation: The proposed design provides the option to specify the percentage of decoy files to use for protection (i.e., the ratio of the decoy files to the original legitimate user files in terms of storage size). The model iteratively creates a decoy file in each run until the total storage size of decoy files surpasses the limit based on the number of votes received (refer to Equation 7). However, unless the decoy file size limit is specified, all the candidate decoy files returned by the decoy file picker module (see Figure 5) are created. Moreover, the designed prototype allows the user to specify the sequence of characters to name decoy files; this is important to avoid false-positive rates. The string provided by the user is combined with the original filename to exhibit a legitimate user file to the attacker. For instance, a candidate decoy file named "report.pdf" can be recreated as "reportdecoy.pdf", or "resecretport.pdf", etc.

As time passes, users can change and modify the list of files, leaving decoy files unaltered; this permits a sophisticated attacker to ignore files and folders that the owner never accesses, nullifying the purpose of deception-based detection. Hence, the generated decoy files could be indistinguishable from legitimate user files to be effective. In RTrap, the proposed design periodically updates the generated decoy files so that the attacker cannot distinguish decoy files from legitimate user files. Obscurement is done by reflecting user activity on decoy files based on a predefined schedule or threshold for triggering the update; whenever the update module is triggered, the update module checks if there is any update to the content of the user file accordingly, updates the decoy files. To avoid scanning the entire system, we can log the file activity and trigger the update only for those directories with the cost of storing file-change statistics. In addition, it is likewise possible to enable dynamic thresholds by learning the timing of file change performed by the user at each point in the day, week, or month and accordingly trigger the decoy-updater.

C. Decoy-Watcher

"RTrap" assumes that users are aware of the decoy files on their machines. Thus, any modifications to the decoy files signal a ransomware presence. The decoy-watcher components of RTrap are responsible for monitoring the planted decoy files. Upon detecting modifications (any changes) to the deployed decoy files, the decoy-watcher will automatically disconnect the host machine from the network and kill the malicious process.

In addition to malicious activity detection, having endpoint protection that autonomously stops or kills the malicious process is the best method of containment during the ransomware attack. This is because *ransomware resembles the behavior of cancer spreading in a body; that is, the longer the ransomware activates in the victim devices, the more harm to the organization as the ransomware can also propagate to the network-attached devices to encrypt as many files as possible*. Toward this, we introduce a decoy-watcher. The decoy-watcher component monitors the generated decoy files in real-time and executes defensive actions such as killing the malicious process or disconnecting the victim from the network whenever decoy files are modified.

In addition to malicious activity detection, endpoint protection that autonomously stops or kills the malicious process is the best containment method during a ransomware attack. Having a mechanism to stop ransomware autonomously is essential because *ransomware resembles the behavior of cancer spreading in a body*; that is, the longer the ransomware activates in the victim devices, the more harm to the organization as the ransomware can also propagate to the network-attached devices to encrypt as many files as possible. Toward this, we introduce a decoy-watcher. The decoy-watcher component monitors the generated decoy files in real-time and executes defensive actions such as killing the malicious process or disconnecting the victim from the network whenever decoy files are modified.

In Windows-based operating systems, the file system monitors any activity performed by the program/process that involves files (e.g., creating, deleting, updating, or moving files). The event (or message) generated by a file system is placed in the operating system's internal buffer. Thus, registering the third-party applications to those events makes it possible to listen for file events and perform application-specific tasks. For instance, Windows Explorer registers for file change events to refresh the list of documents inside the

directory. Likewise, the proposed decoy watcher is registered with the file system to get a notification whenever the decoy files are changed, renamed, created, or deleted. File changes, rename, create and delete operations are chosen because ransomware can (1) change and rename a list of files (i.e., encryption and renaming of file with specific file extension) and (2) create and delete files (i.e., ransomware encrypts the original file to create an encrypted file and then deletes the original file). Thus, upon receiving the notification, the decoy-watcher can take all the necessary actions to mitigate the attack; for example, the decoy-watcher can disconnect the host from the network, kill the malicious process, clear all the registry values, and shut down the system. Furthermore, with the cost of more file losses, extending the "mitigation task" to notify the system admin by SMS or email before the system shutdown is possible.

Overall, the integration of decoy-watcher with a file system makes it efficient in terms of real-time memory usage and CPU usage statistics. *In this work, we implement the proposed "RTrap, including the decoy-watcher, on "Windows" operating systems because Windows is a widely-used operating system in the world.*

Integrating the decoy-watcher with a file system makes real-time memory usage and CPU usage statistics efficient. *In this work, we implement the proposed "RTrap," including the decoy-watcher, on "Windows" operating systems because Windows is a widely-used operating system worldwide.*

V. Experiment Setup and Analysis

This section evaluates the performance of RTrap with actual ransomware and document samples on real operating systems. The experiment results are discussed in terms of "file-loss analysis" (i.e., the number of files lost before detection), "stopping distance"(i.e., the time it takes to contain the ransomware activity), "sustainability analysis," "false-positive rate," and "overall impact" on the target system in terms of resource consumption.

A. Experiment Setup

To assess the performance of RTrap, first, we implemented RTrap with all the experiments on Windows 10 and built a set of representative ransomware samples and user documents. The ransomware families used for the experiment were Babuk, BlackRuby, CobraLocker, CoronaVirus, Crisis/Dharma, Egregor, GlobeImposter, Grandcrab, Jigsaw, LockerGoga, Makob, Maze, Petya, Phobos, Polyransom/Virlock, Ryuk, Shade, Sodinokibi, TeslaCrypt, and Xorist. These ransomware families are the most active (or top) ransomware attacks from 2018 to Q1 of 2020 [39]. For each family, we collected 15 executable samples (if available) from different online malware archives, such as VirusShare [40], to constitute 1106 samples. Finally, we built a set of representative files for ransomware to attack. The representative user documents in this study were collected from two publicly available corpora, i.e., govdocs1 [41] and Coldwell's audio files [42]. These records constituted 10311 files, including image files, spreadsheets, programming source codes, reports, pdf, recordings, music, archives, and so forth. Each record was randomly assigned to a folder and subfolder (see Figure 5) and placed in a virtual machine (VM) with Windows 10 installed. After each run of ransomware, a VM was reset to an earlier snapshot (1) to avoid the impact of the previous ransomware execution on the current execution and (2) also to make rerunning the analysis of ransomware activity much more straightforward. Since some ransomware variants have an anti-analysis feature, we set a timeout time to 20 minutes to avoid a long waiting time for pointless ransomware samples. Anti-analysis exist because out of the 1106 ransomware variants we executed, only 846 of them get activated in 20 minutes.

For comparison, we implemented two related works that employ deception-based ransomware detection (i.e., Lee et al. [22] and Rwguard [25]) and three dynamic-behavior analysis methods (i.e., EGB [18], DeepGuard [19], and File-entropy [21]) (see Section II-B for details). Note that since the source code of Lee et al. [22], File-entropy [21], and Rwguard [25] are not available to the public, they are re-implemented for comparison. For the EGB [18] model, the source code and training data are publicly available. The source code and training data of DeepGuard [19] are shared with us by the authors of DeepGuard [19]. The same experimental setup was used to investigate RTrap and all the baseline models.

B. Decoy File Quality Metric

Let us denote ransomware as R and a set of legitimate user files as F. The quality of deceptive files created via a decoy file generation method g is given as follows [43]:

$$\Pr[|X_R^g(F) = n|] \tag{8}$$

where $X_R^g(F)$ is the variable that outputs the total number of legitimate user files that a ransomware R encrypts before attacking one in the set of deployed decoy files. For instance, if there are 100 user files, and ransomware R scrambles 40 user files before getting to any decoy files during the attack, the value of n in Equation 8 becomes 40. This result suggests that the deployed decoy file ensures a 60% deception rate (or detection rate) against the ransomware R. However, $n = 0$ indicates that the deployed decoy file promptly deceives the ransomware. Hence, a good deception strategy should minimize the likelihood that ransomware scrambles legitimate user files before being detected.

To apply this decoy file quality metric, we develop a simple analysis tool that logs every interaction between ransomware and user files. This tool adds the event data to a new line, where each line in the log file comprises four fields, which are "timestamp," "document path," "event type," and "counter", where a counter is a variable that increments from 0 to n whenever legitimate user files are changed by ransomware. To make the log file safer, we save it as an executable file (.exe file). We refer to this metric as *file-loss analysis* in the subsequent section as it quantifies the number of files lost before ransomware R encrypts one in the deployed decoy files.

In both the EGB and DeepGuard, ransomware detection is modeled as a classification problem. Hence, to have a

fair comparison with the deception-based approach, for the EGB and DeepGuard experiments, we count the number of files encrypted by the ransomware before the model outputs the current activity as the ransomware activity. Whenever the model outputs the current activity as the ransomware activity, we stop writing to the log file and count the number of files modified by the ransomware during that experiment (i.e., from the log file).

C. File-Loss Analysis

This Section assesses the quality of the proposed decoy file generator in deceiving ransomware. As discussed in Section V-B, the amount of data lost before detection (i.e., before decoy files are accessed) is our valuable metric for quality measurement because when the ransomware attacks the decoy files, the decoy-watcher in RTrap quickly fires the mitigation task to stop the malicious process from protecting most files from the ransomware. Conversely, when the decoy files are accessed lately by the ransomware, most user files will be scrambled/encrypted; this leads to more ransom demand. The experiment results in Figure 6 indicate the average file loss (y-axis) for each ransomware family used in the experiment.

The results show that all the deception algorithms effectively deceive the attack at an early stage with a minimal file loss (on average, 200, 288, 296, 273, 163, and 18, for Lee et al. [22], Rwguard [25], DeepGuard [19], EGB [18], File-entropy [21] and RTrap, respectively), considering that experiments have been carried out using the emerging and more sophisticated ransomware families and variants. However, unlike RTrap, the other baseline models' deception quality (or detection performance) fluctuates significantly as the ransomware family changes. The number of lost files before deception (or attacking decoy files) ranges from 107 to 345 for Lee et al. [22] strategy, from 12 to 527 for File-entropy [21], and it ranges from 152 to 486 for Rwguard [25]. Similarly, the number of lost files for DeepGuard [19] and EGB [18] fluctuates between 106 and 596.

The main reason for the disparity and increase in the number of lost files across ransomware families for Rwguard [25] and Lee et al. [22] is that they create decoy files without considering the property of files in a directory and the ransomware file prioritization strategy. As the ransomware strategies vary in different contexts (see Section III), planting decoy files heuristically cannot guarantee the model's effectiveness. Subsequently, this behavior makes the model highly sensitive to the ransomware family or variant performing the attack. DeepGuard [19] and EGB [18] rely on the threshold (where the threshold is learned from examples during the training) to dictate that the current activity is a ransomware activity or benign activity. For instance, in EGB, any running process with a trust score higher than the maximum threshold value will be addressed as malicious activity. However, this approach is ineffective for ransomware families that employ process splitting, and hence the performance of EGB and DeepGuard fluctuates with ransomware families.

One of the constraints of the deception-based approach is that they have a narrow field of view; that is, they only see the access directly toward the decoy files. Subsequently, when the ransomware file-type/file-size prioritization is not similar to that of the deployed decoy file attributes, the decoy-watcher may never detect the ransomware (or it may detect it lately) after several files are encrypted. A decoy file generation strategy that is adaptive or resilient to any file ordering or prioritization technique utilized by ransomware is one of the missing components in the existing literature. In Rwguard [25], the authors suggest creating decoy files with specific file types and sizes. However, creating decoy files in this manner can lead to late discovery (or no detection) for some ransomware families that encrypt any types of files (or file types other than the decoy files). Consequently, the performance of Rwguard is unstable across ransomware families concerning the file-loss analysis.

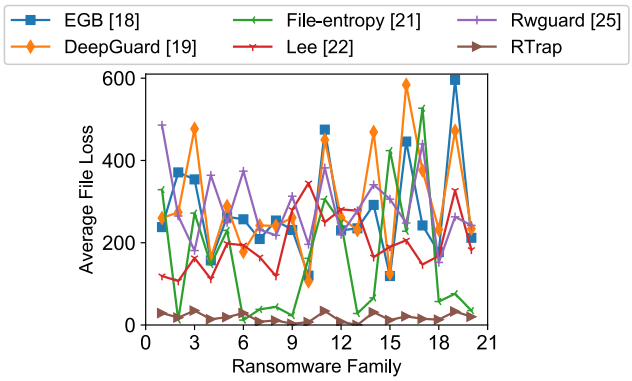


Fig. 6. File-loss Analysis (The x-axis denotes the ransomware family, sorted by family name where x=1 indicates Babuk, whereas x=20 denotes the Xorist family).

In contrast, the proposed model significantly and consistently outperformed the other approaches in deceiving/detecting different ransomware families with an average of 18 lost files per 10311 user files; this is equal to a 99.82% deception rate. The RTrap performance likely emanates from the proposed decoy file generation strategy's data-driven solution. As discussed in Section IV-B, in RTrap, the decoy files are generated intelligently with a data-driven solution where each file in a directory votes for its corresponding representative decoy file after iteratively exchanging attractiveness messages. This data-driven strategy allows the model to decide and select the required number of files without manually specifying the number of decoy files. Creating decoy files in this manner permits the model to deceive the attack by being accessed at an earlier stage. Hence, the variety in file-sorting/prioritization techniques utilized by ransomware has little impact on RTrap deception quality. Overall, the file-loss analyses highlight that the data-driven nature of the proposed decoy file generation strategy resolves the issue of ransomware variation in some way and provides better deception quality than the two baseline models across ransomware families.

Figure 6 also depicts how effectively the investigated models detect ransomware families that employ multithreading. Our experiment includes ransomware that parallelizes the encryption process, such as LockerGoga (number 10 in Figure 6) and Ryuk (number 16 in Figure 6). The result depicted in Figure 6 shows that RTrap can minimize the file loss caused

by LockerGoga and Ryuk ransomware by 251, 267, 124, 171, and 181on average compared to Lee, Rwguard, DeepGuard, EGB, and File-entropy respectively.

In addition, to assess the impact of the variation of file types and folder structure on the model detection performance, we repeat the experiment using the dataset collected from Microsoft Research Open Data [44]. Nearly 320 Gigabytes of files with different file types such as CSV, TXT, image, audio, JSON, pdf, ppt, HTML, cs, m, mat, bvh, and Docx are selected and planted on the test virtual machine. All tests use the same hardware, Operating System, and ransomware reported in Figure 6. As the file types, file content, folder structure, and file quantity change, one expects to see an increased variation in performance, and the results support this. However, the performance of RTrap is more consistent with the one reported in Figure 6, affirming the adaptive nature of the decoy file generation strategy.

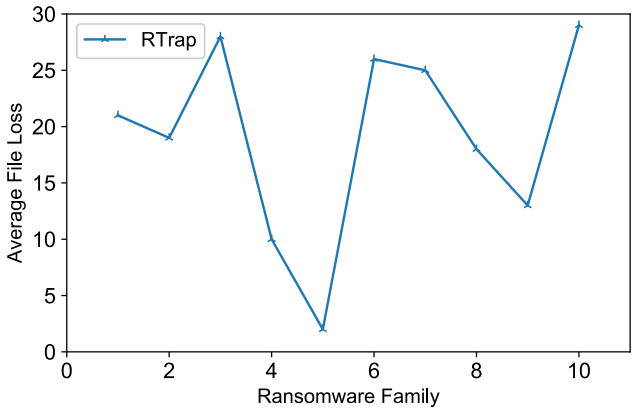


Fig. 7. Sustainability Analysis of RTrap (The x-axis denotes the ransomware family, sorted by family name where x=1 indicates AvosLocker, whereas x=10 denotes the Suncrypts family).

D. Sustainability Analysis of RTrap

Numerous malware defense solutions have been developed, yet research in [45] and [46] indicates that many quickly became outdated due to the fast evolution of benign and malware program development. Cybersecurity today feels too much like a game of cat and mouse. As the security experts invent a novel attack detection mechanism, criminals will concoct another strategy to go unnoticed. The occurrence of zero-day malware/ransomware is a fundamental and progressing issue for all organizations.

As discussed in Section II-B, numerous malware detection solution exists. Rule-based or Machine learning-based solution identifies malware by predicting a given app as benign or malicious based on the extracted static (e.g., OpCodes), dynamic (system API calls), or hybrid features. However, due to the frequent benign program and malware evolution, malware classifiers built on these features may not be sustainable as they require retraining to cope with the evolution of benign and malware programs; otherwise, the detection performance deteriorates [45], [46], [47], [48], [49].

The sustainability challenge with existing malware detection has been a point of interest in recent years [45], [46], [50]. For instance, authors in [46] discovered that the performance of the state-of-the-art Android malware detector solution drops by 60% within one year. From these recent prior works, developing a sustainable ransomware detector is pivotal; if not, without retraining, the detector will not have the option to recognize new ransomware, which presently continues to emerge and surge.

As pointed out in the different sections of this paper, RTrap provides signature-less and behavior-agnostic ransomware detection. *Thus, unlike solutions that rely on the feature extracted from benign programs and malware, the RTrap program is sustainable concerning the benign program evolutions.*

While the deception quality of RTrap in the original test ransomware was nearly 99.82%, we can not tell how the RTrap detection performance will change over time as new ransomware gets released. Toward this, we conducted an experiment using ten ransomware families released to the VirusShare [40] between 2021 and Q1 of 2022. The ransomware families investigated for the RTrap sustainability studies are AvosLocker, BlackByte, BlackCat, Clop, Conti, Cuba, HiveLeaks, Lockbit, Snatch, and Suncrypts. The same file setup as in Section V-A was used to assess the sustainability of RTrap. Our preliminary result in Figure 7 shows that RTrap is sustainable at reasonable costs, with an average of 20 files lost per 10311 user files (i.e., on average, two more files get encrypted compared to the upshot reported in Figure 6). This result also means the deception rate of RTrap decays only by 0.02% (i.e., 99.82% - 99.80%). The main takeaway of this experiment is that RTrap generates a decoy file adaptively without considering the static or dynamic property of the ransomware. Subsequently, it is agnostic to ransomware variants and sustainable over time.

E. Decoy-Watcher Analysis

As discussed in Section IV-C, upon detecting malicious activity, the ransomware incident containment strategies introduced in the proposed decoy-watcher perform two main activities: (1) disconnecting the host from the network by disabling the internet connectivity and (2) killing the malicious process to minimize the risk of the ransomware continuing the encryption process. This setting can likewise be configured to promptly turn off the victim device based on the level of risk acceptable to the organization (or organizational policy). The inability to rapidly disconnect the victim from the system may add to the incident by permitting the ransomware to continue encrypting the user files on the local system and network-attached devices so that the recovery endeavor returns the system to the original state is increased. Thus, it is important to contain the ransomware quickly.

In RTrap, the decoy-watcher component will promptly execute the containment decision to disconnect the victim from the network and stop malicious process execution. However, as discussed in Section III, ransomware can employ multi-threading, process splitting, or offloading process strategies to execute its malicious task. Thus, halting the malicious process or stopping the ransomware execution takes time. In order

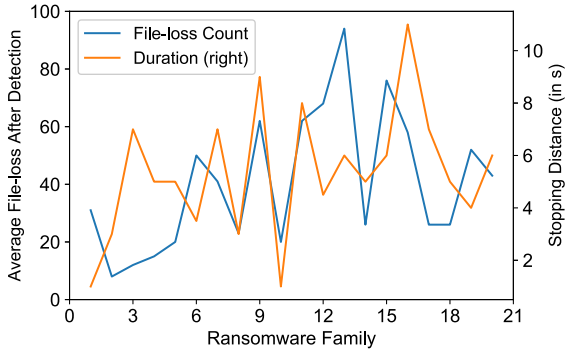


Fig. 8. The x-axis denotes the ransomware family, sorted by family name where x=1 indicates Babuk, whereas x=20 denotes the Xorist family.

to access the effectiveness of the decoy-watcher in terms of containment of the attack, *we introduce a new metric called "stopping distance" (SD)*, which is the time it takes for the decoy-watcher to stop the execution of the ransomware attack after receiving the first notification from the file system. Mathematically SD is defined as follows:

$$SD = T^E\_end - T^E_{notifiation} \tag{9}$$

In other words, Equation 9 can be read as the time difference between the first time the decoy file is accessed and the last moment the encryption completely stops. The experiment result shown in Figure 8 was calculated from the timestamp information stored in a log file. In Figure 8, the x-axis indicates the ransomware family, the left-side y-axis denotes the average number of files lost after detection, and the right-side y-axis signifies the stopping distance in seconds. The outcome reveals that the average time to stop the ransomware process execution after receiving the first notification is 5.35 s on average, and this value increases to 11 s for ransomware family (e.g., Ryuk) that utilizes more sophisticated techniques. Meanwhile, the average number of files lost after the first deception of the ransomware to final termination is 41. Hence, considering both the average number of lost files to deceive the attack (i.e., 18) and to contain/stop the attack (i.e., 41), RTrap can guarantee a 99.42% (i.e., 10252/10311) protection rate against the ransomware variants utilized in this study. Note that 10311 is the total number of legitimate user files, and 10252 is the number of user files not altered by ransomware after the attack (i.e., $10311 - 41 - 18 = 10252$).

Overall, the disparity in the value of SD attributes to the variation in the number of a malicious process spawned by the ransomware family or variant. For the ransomware families that employ multithreading or uses multiple processes to perform the encryption, the value of SD is high, and thus more files will be encrypted; when the ransomware employs a single process, the proposed decoy-watcher takes less time to stop its execution. Note that the value of SD can be reduced to less than 2 seconds if the mitigation task was programmed to turn off the victim device.

F. Analysis of RTrap False Positive Rate

This experiment investigates the false-positive rate (FPR) (refer to 10) of RTrap by counting the number of times legitimate users or programs mistakenly access the deployed deco-files. The false-positive analysis outlined in [51] is adopted for the experiment. We develop a simple file watcher that logs every access to the decoy file and records it in a hidden location. Whenever the decoy file gets accessed, it will add a new line with a path to the accessed decoy file and a timestamp to the log file.

$$FPR = \frac{FP}{FP + TN} \tag{10}$$

FP in Equation 10 indicates the number of False positives (i.e., the number of decoy files accessed by participants), and TN indicates True Negatives (i.e., the total number of non-decoy files). First, we randomly select ten administrative staff of the university. Participants are informed and asked to run the *RTrap* decoy-file generators on their machines. Before generating the decoy files, participants are asked to provide an easily recognizable tag (i.e., it can be a prefix or suffix) for naming a decoy file while also making it alluring to the attacker. We believe configuring the decoy-file name in such a way would make the file easily recognizable as a decoy file by the participant but could make it difficult for the attacker to differentiate it.

Intuitively, decoy files planted on a participant's computer for ransomware detection are not supposed to be accessed by a legitimate user. Because decoy files are placed there to attract adversaries to open and modify them, any accidental access to decoy files by the participants or legitimate software is a false positive.

Following the decoy-file generation, the participants were asked to install the file watcher on their system to record user activity. Note that we also informed the participants that they were being monitored. Then, the log file was used to measure the number of times the decoy files were accessed within four weeks of the experiment. On day one of the experiment, the file watcher was configured to generate an alert to allow participants to learn and memorize the decoy files. From the second day on, any access to the decoy files will be recorded secretly without generating an alert.

At the end of the experiment, the log file from the participant's computer is collected, and our findings are summarized in Table I. While more longitudinal research is expected to generalize the correlation between the quantity of planted decoy-files, and their false-positive rate analysis, our preliminary outcomes indicate that *RTrap* decoy-file generation strategy system has an average false-positive rate of 0.2%. As shown in Table I, the highest access to the decoy-files was assessed observed in *PC10* (i.e., 10^{th} participant) which is also the computer with the highest number of legitimate user files and the highest number of decoy-files.

In addition to providing adequate training for the user, it is feasible to lower the false-positive rate by hiding the visibility of decoy files. We also did not encounter any ransomware that ignores hidden files; nevertheless, targeted malware may do so to avoid detection.

TABLE I
FALSE POSITIVE ANALYSIS OF RTRAP

Computer	#of user files	#of placed decoy-files	#of decoy-file accessed
PC1	1793	35	1
PC2	658	20	0
PC3	884	17	2
PC4	1735	52	6
PC5	3596	121	0
PC6	1465	34	6
PC7	624	12	5
PC8	1019	25	3
PC9	990	27	1
PC10	5607	204	12
Total	18371	547	36

Overall, finding a false-positive rate of less than 1% is encouraging, but it might be further reduced with better decoy-file placement strategies. For instance, an interested researcher might devise a decoy file placement system that considers both program behavior and user behaviors to lower false-positive rates.

G. Performance Analysis

As the final metric to evaluate the practicability of the proposed method, we assess the performance of RTrap, in terms of "runtime overhead." First, we assess the decoy file selection module's runtime memory requirement and time complexity. Following that, we conduct the runtime analysis for the decoy-watcher module.

The memory utilization for the execution of the proposed decoy file generation algorithm is primarily to store the value of three matrices: $Sim(i, j)$, $Resp(i, j)$, and $Avail(i, j)$ (refer to Section IV-B.3). Suppose that each element of the matrix requires 4 bytes of memory, and the total number of the data points in a directory is N; the total memory required to store the three matrices is at least $3 \times N^2 \times 4$ bytes. In RTrap, the decoy file selection process involves hierarchical execution; that is, decoy files are selected for the root directory first and moves to the subdirectories one by one. The utilization of this strategy enables the efficient execution of the computation without loading the entire data points from the storage devices to the main memory. Hence, the runtime memory requirement of the proposed decoy file generation strategy is minimal. With regards to time complexity, the entire update process of $Resp(i, j)$ and $Avail(i, j)$ requires $O(TN^2)$ time, where T represents the number of iteration required to converge. *Note that the decoy file generation process is executed just a single time for the whole system, and the procedure repeats for specific directories based on the scheduling policy to make the update to the decoy files*. Thus, the online/runtime overhead is minimal.

As discussed in Section IV-C, our decoy-watcher is registered with the file system to get informed promptly whenever a decoy file is accessed. In the proposed decoy-watcher, monitoring directories for changes to decoy files are implemented with a "file name filter", which permits the event to raise only when the name of the file changed matches the filter condition (i.e., only decoy files). This has the advantage of reducing the buffer usage of the real-time monitoring of decoy files. *Importantly, as the "file monitoring task" is the default task performed by the file system, there is no significant computational overhead introduced by the proposed RTrap*. The only exception is when a specific event related to the decoy file is received (i.e., during the ransomware attack). At that time, a single event can take up 16 bytes in the buffer in addition to the bytes required to store the name of the decoy file currently accessed. In the worst case, if the event was raised for a decoy file with a file name with 255 Unicode characters (i.e., maximum allowed number of characters in NTFS), it takes up to 526 bytes of memory (i.e., two bytes per character), which is negligible.

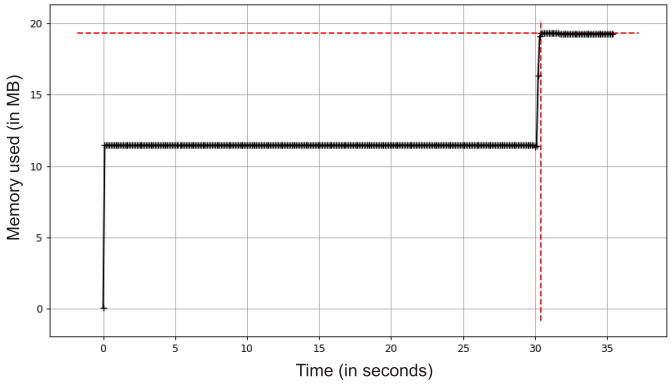


Fig. 9. Run-time analysis of decoy-watcher.

To show the real-time memory usage of the proposed decoy-watcher, we conduct the memory trace experiment to measure the amount of memory currently in use for a decoy-watcher process and child-process. The runtime memory usage of the proposed decoy-watcher is shown in Figure 9. The first part in the initial 30 seconds of Figure 9 shows that the decoy-watcher is running, but no suspicious activity is detected. On the other hand, the peak memory usage was recorded (i.e., 20 MB) at the time when the decoy files are accessed (i.e., access to decoy files), and the decoy-watcher was taking a mitigation task. Overall, the amount of memory that is being consumed by the decoy-watcher module is less than 20 MB, which is insignificant compared with antiviral solutions that use hundreds of megabytes. During the dynamic analysis, antiviral solutions perform memory-intensive tasks (e.g., intercepting and correlating the API call sequence with malware signatures) and track each process activity for digital evidence (e.g., dropped executable with millions of malware signatures and observing changes to system registry activity [52]); thus, antiviral solutions demand a significant amount of RAM). Overall, the run-time resource usage of the proposed RTrap has negligible runtime overhead.

VI. CONCLUSION

Although prevention is ideal for any attack, not all ransomware attacks can be prevented due to criminals' signature obfuscation and zero-day exploitation strategies. In recognizing that attackers will succeed one day in bypassing the prevention strategy utilized by an enterprise, this work proposes a systematic strategy, called "RTrap," to create deceptive files via machine learning to lure the attacker (or ransomware) into accessing it. Upon detecting any potential access to deceptive

files, the proposed RTrap model autonomously contains the incidence by disconnecting the victim from the network and killing all the malicious processes. The realistic experiment using emerging ransomware families shows that RTrap can detect the ransomware with an average of 18 lost files per 10311 legitimate user files. The experiment result also shows that the proposed solution provides a post-breach defense agnostic to the attack's ransomware family or variant. Besides, on average, the proposed lightweight decoy-watcher can trap, contain, and control the execution of the ransomware in less than 5.35 s. Hence, we consider the proposed approach practical and efficient for post-breach detection and containment. Moreover, this work also lays out the main technical concerns of emerging ransomware families and shortfalls of existing works, which are vital for other researchers and technologists working on the related issues. Introducing an automated strategy to detect ransomware before it starts the encryption and designing mechanisms to auto-detect and remove ransomware keys from the registry is the part we left for future work.

REFERENCES

[1] B. Zhang, W. Xiao, X. Xiao, A. K. Sangaiah, W. Zhang, and J. Zhang, "Ransomware classification using patch-based CNN and self-attention network on embedded N-grams of opcodes," *Future Gener. Comput. Syst.*, vol. 110, pp. 708–720, Sep. 2020.

[2] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proc. 23rd Annu. Comput. Secur. Appl. Conf. (ACSAC)*, Dec. 2007, pp. 421–430.

[3] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," in *Proc. 5th Int. Conf. Malicious Unwanted Softw.*, Oct. 2010, pp. 23–30.

[4] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A large-scale, automated approach to detecting ransomware," in *Proc. 25th USENIX Secur. Symp. (USENIX Security)*, 2016, pp. 757–772.

[5] F. De Gaspari, D. Hitaj, G. Pagnotta, L. De Carli, and L. V. Mancini, "The naked sun: Malicious cooperation between benign-looking processes," 2019, *arXiv:1911.02423*.

[6] R. Samani and C. Beek, "McAfee labs threats report," McAfee, Santa Clara, CA, USA, Tech. Rep., Aug. 2019.

[7] B. Whitham, "Automating the generation of enticing text content for high-interaction honeyfiles," in *Proc. 50th Hawaii Int. Conf. Syst. Sci.*, 2017, pp. 1–10.

[8] Emsisoft Malware Lab. *The State of Ransomware in the US: Report and Statistics 2019*. Accessed: Jan. 20, 2020. [Online]. Available: https://blog.emsisoft.com/en/34822/the-state-of-ransomware-in-the-us-report-and-statistics-2019/

[9] Sophos-Lab. (Jan. 2018). *The State of Endpoint Security Today*. [Online]. Available: https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/endpoint-survey-report.pdf

[10] S. Poudyal, K. P. Subedi, and D. Dasgupta, "A framework for analyzing ransomware using machine learning," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Nov. 2018, pp. 1692–1699.

[11] R. M. A. Molina, S. Torabi, K. Sarieddine, E. Bou-Harb, N. Bouguila, and C. Assi, "On ransomware family attribution using pre-attack paranoia activities," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 1, pp. 19–36, Mar. 2022.

[12] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop it): Stopping ransomware attacks on user data," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2016, pp. 303–312.

[13] K. Cabaj and W. Mazurczyk, "Using software-defined networking for ransomware mitigation: The case of cryptowall," *IEEE Netw.*, vol. 30, no. 6, pp. 14–20, Nov./Dec. 2016.

[14] O. M. Alhawi, J. Baldwin, and A. Dehghantanha, "Leveraging machine learning techniques for windows ransomware network traffic detection," in *Cyber Threat Intelligence*. New York, NY, USA: Springer, 2018, pp. 93–106.

[15] A. O. Almashhadani, M. Kaiiali, S. Sezer, and P. O'Kane, "A multi-classifier network-based crypto ransomware detection system: A case study of Locky ransomware," *IEEE Access*, vol. 7, pp. 47053–47067, 2019.

[16] S. Sharmeen, Y. A. Ahmed, S. Huda, B. S. Kocer, and M. M. Hassan, "Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches," *IEEE Access*, vol. 8, pp. 24522–24534, 2020.

[17] M. Alam, S. Bhattacharya, S. Dutta, S. Sinha, D. Mukhopadhyay, and A. Chattopadhyay, "RATAFIA: Ransomware analysis using time and frequency informed autoencoders," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2019, pp. 218–227.

[18] S. Aurangzeb, R. N. B. Rais, M. Aleem, M. A. Islam, and M. A. Iqbal, "On the classification of microsoft-windows ransomware using hardware profile," *PeerJ Comput. Sci.*, vol. 7, p. e361, Feb. 2021.

[19] G. O. Ganfure, C.-F. Wu, Y.-H. Chang, and W.-K. Shih, "DeepGuard: Deep generative user-behavior analytics for ransomware detection," in *Proc. IEEE Int. Conf. Intell. Secur. Informat. (ISI)*, Nov. 2020, pp. 1–6.

[20] S. Baek, Y. Jung, D. Mohaisen, S. Lee, and D. Nyang, "SSD-assisted ransomware detection and data recovery techniques," *IEEE Trans. Comput.*, vol. 70, no. 10, pp. 1762–1776, Oct. 2020.

[21] C.-M. Hsu, C.-C. Yang, H.-H. Cheng, P. E. Setiasabda, and J.-S. Leu, "Enhancing file entropy analysis to improve machine learning detection rate of ransomware," *IEEE Access*, vol. 9, pp. 138345–138351, 2021.

[22] J. Lee, J. Lee, and J. Hong, "How to make efficient decoy files for ransomware detection?" in *Proc. Int. Conf. Res. Adapt. Convergent Syst.*, Sep. 2017, pp. 208–212.

[23] J. A. Gómez-Hernández, L. Álvarez-González, and P. García-Teodoro, "R-locker: Thwarting ransomware action through a honeyfile-based approach," *Comput. Secur.*, vol. 73, pp. 389–398, Mar. 2018.

[24] Watchpoint Data. (2018). *Cryptostopper*. [Online]. Available: https://www.watchpointdata.com/cryptostopper

[25] S. Mehnaz, A. Mudgerikar, and E. Bertino, "RWGuard: A real-time detection system against cryptographic ransomware," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. New York, NY, USA: Springer, 2018, pp. 114–136.

[26] G. Hull, H. John, and B. Arief, "Ransomware deployment methods and analysis: Views from a predictive model and human responses," *Crime Sci.*, vol. 8, no. 1, p. 2, Dec. 2019.

[27] Sophos Lab. (2020). *Sophos Lab Security Threat Report*. [Online]. Available: https://www.sophos.com/en-us/labs/security-threat-report.aspx

[28] K. Sudhakar and S. Kumar, "An emerging threat fileless malware: A survey and research challenges," *Cybersecurity*, vol. 3, no. 1, pp. 1–12, Dec. 2020.

[29] *A Sophoslabs White Paper 2019*. Accessed: Dec. 21, 2019. [Online]. Available: https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophoslabs-ransomware-behavior-report.pdf

[30] (2019). *Black Ruby: Combining Ransomware and Coin Miner Malware*. [Online]. Available: https://www.acronis.com/en-us/blog/posts/black-ruby-combining-ransomware-and-coin-miner-malware

[31] AV-TEST. (Mar. 2020). *Malware Statistics and Trends Report: AV-Test*. [Online]. Available: https://www.av-test.org/en/statistics

[32] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, Feb. 2007.

[33] E. Bisong, "Introduction to scikit-learn," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. CA, USA: Springer, 2019, pp. 215–229.

[34] A. van Wyk, "Encoding cyclical features for deep learning," EPI-USE Lab, Pretoria, South Africa, Tech. Rep., 2018.

[35] T. Ronan, Z. Qi, and K. M. Naegle, "Avoiding common pitfalls when clustering biological data," *Sci. Signaling*, vol. 9, no. 432, p. re6, Jun. 2016.

[36] Y. Xu, D. Zhang, and J.-Y. Yang, "A feature extraction method for use with bimodal biometrics," *Pattern Recognit.*, vol. 43, no. 3, pp. 1106–1115, Mar. 2010.

[37] L. Zhang, R. Lukac, X. Wu, and D. Zhang, "PCA-based spatially adaptive denoising of CFA images for single-sensor digital cameras," *IEEE Trans. Image Process.*, vol. 18, no. 4, pp. 797–812, Apr. 2009.

[38] H. Ashtiani, S. Kushagra, and S. Ben-David, "Clustering with same-cluster queries," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3216–3224.

[39] Kaspersky-Lab. (May 2020). *Ransomware 2018-2020*. [Online]. Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/100/2020/05/12075747/KSN-article_Ransomware-in-2018-2020-1.pdf

[40] J.-M. Roberts. (2011). *Virus Share*. [Online]. Available: https://virusshare.com
[41] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *Digit. Invest.*, vol. 6, pp. S2–S11, Sep. 2009.
[42] N. Coldwell. *Comparison of Audio Compression*. Accessed: Mar. 5, 2019. [Online]. Available: http://nigelcoldwell.co.uk/audio/index.htm
[43] Z. A. Genç, G. Lenzini, and D. Sgandurra, "On deception-based protection against cryptographic ransomware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. New York, NY, USA: Springer, 2019, pp. 219–239.
[44] *Microsoft Research Open Data*. Accessed: Mar. 21, 2022. [Online]. Available: https://msropendata.com/
[45] H. Cai and J. Jenkins, "Towards sustainable Android malware detection," in *Proc. 40th Int. Conf. Softw. Eng., Companion Proc.*, May 2018, pp. 350–351.
[46] X. Fu and H. Cai, "On the deterioration of learning-based malware detectors for android," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May 2019, pp. 272–273.
[47] H. Cai, X. Fu, and A. Hamou-Lhadj, "A study of run-time behavioral evolution of benign versus malicious apps in Android," *Inf. Softw. Technol.*, vol. 122, Jun. 2020, Art. no. 106291.
[48] H. Cai, "Embracing mobile app evolution via continuous ecosystem mining and characterization," in *Proc. IEEE/ACM 7th Int. Conf. Mobile Softw. Eng. Syst.*, Jul. 2020, pp. 31–35.
[49] W. Li, X. Fu, and H. Cai, "AndroCT: Ten years of app call traces in android," in *Proc. IEEE/ACM 18th Int. Conf. Mining Softw. Repositories (MSR)*, May 2021, pp. 570–574.
[50] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 2, pp. 1–28, Apr. 2020.
[51] M. B. Salem and S. J. Stolfo, "Decoy document deployment for effective masquerade attack detection," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Heidelberg, Germany: Springer, 2011, pp. 35–54.
[52] C. H. Malin, E. Casey, and J. M. Aquilina, *Malware Forensics Field Guide for Windows Systems: Digital Forensics Field Guides*. Amsterdam, The Netherlands: Elsevier, 2012.

Gaddisa Olani Ganfure (Member, IEEE) received the bachelor's degree in computer science and IT from Wollega University, Ethiopia, in 2010, the master's degree in computer science from Addis Ababa University in 2013, and the Ph.D. degree from the Social Network Analysis and Human-Centered Computing Department, National Tsing Hua University, and Academia Sinica, Taiwan, in 2020. From September 2010 to July 2017, he was a Lecturer with the Department of Computer Science, Dire Dawa University, Dire Dawa, Ethiopia. He is currently an Assistant Professor of computer science with the School of Computing, Department of Computer Science, Dire Dawa University. He was awarded a scholarship by Taiwan International Graduate Program (TIGP) to pursue his Ph.D. degree at NTHU. His research interests include big data analysis, cybersecurity, AI-based intrusion detection systems, natural language processing, and user behavior modeling for cyber deceptions.

Chun-Feng Wu (Member, IEEE) received the M.S. degree from the Department of Computer Science, National Tsing Hua University, in 2016, and the Ph.D. degree from the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, in 2021. Currently, he is an Assistant Professor with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan. Previously, he was a Post-Doctoral Scholar at the Department of Computer Science, Harvard University, Cambridge, MA, USA, from 2021 to 2022. He worked in research and development alternative service at the Institute of Information Science, Academia Sinica, Taipei, from 2017 to 2021. His primary research interests include memory/storage systems, embedded systems, operating systems, and the next-generation memory/storage architecture designs.

Yuan-Hao Chang (Fellow, IEEE) received the Ph.D. degree in computer science from the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. He is currently a Research Fellow with the Institute of Information Science, Academia Sinica, Taipei, where he worked as an Associate Research Fellow from March 2015 to June 2018 and an Assistant Research Fellow from August 2011 to March 2015. His research interests include memory/storage systems, operating systems, embedded systems, and real-time systems. He is a Senior Member of ACM.

Wei-Kuan Shih (Member, IEEE) received the B.S. and M.S. degrees in computer science from the National Taiwan University and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign. From 1986 to 1988, he was with the Institute of Information Science, Academia Sinica, Taiwan. He is currently a Professor with the Department of Computer Science, National Tsing Hua University, Taiwan. He has published more than 130 papers in professional journals and conferences. His research interests include real-time systems, wireless sensor networks, distributed file systems, embedded file systems, and energy issues pertaining to cloud computing.