Peeler: Profiling Kernel-Level Events to Detect Ransomware

Muhammad Ejaz Ahmed[1(✉)], Hyoungshick Kim[2], Seyit Camtepe[1], and Surya Nepal[1]

[1] Data61 CSIRO, Marsfield, NSW 2122, Australia
{ejaz.ahmed,seyit.camtepe,surya.nepal}@data61.csiro.au
[2] Sungkyunkwan University, Suwon, South Korea
hyoung@skku.edu

Abstract. Because the recent ransomware families are becoming progressively more advanced, it is challenging to detect ransomware using static features only. However, their behaviors are still more generic and universal to analyze due to their inherent goals and functions. Therefore, we can capture their behaviors by monitoring their system-level activities on files and processes. In this paper, we present a novel ransomware detection system called "Peeler" (**P**rofiling k**E**rn**E**l -**L**evel **E**vents to detect **R**ansomware). Peeler first identifies ransomware's inherent behavioral characteristics such as stealth operations performed during the attack, processes execution patterns, and correlations among different kernel-level events by analysing a large-scaled OS-level provenance data collected from a diverse set of ransomware families. Peeler specifically uses a novel NLP-based deep learning model to fingerprint the contextual behavior of applications by leveraging Bidirectional Encoder Representations from Transformers (BERT) pre-trained model. We evaluate Peeler on a large ransomware dataset including 67 ransomware families and demonstrate that it achieves a 99.5% F1-score.

Keywords: Fileless malware · Ransomware detection · Deep learning · Screen-locker · Malware behavior analysis · Machine learning

1 Introduction

Ransomware is a growing threat that typically operates by either encrypting a victim's files (called *crypto* ransomware) or locking the victim's desktop screen (termed as *screen-locker* ransomware) until the victim pays a ransom. Recent statistics predict that there will be a ransomware attack every 11 s by 2021 – the global cost will be $20 billion yearly [2]. In 2019, the U.S. alone was hit by an unprecedented barrage of ransomware attacks that impacted more than 966 government agencies, educational institutions, and healthcare providers at a potential cost of around $7.5 billion [25]. As a result, a large number of proposals have recently been proposed to fight against ransomware as follows: machine learning models (e.g., [12,18,22,33,34]), use of decoy files (e.g., [12,14,27]), use

E. Bertino et al. (Eds.): ESORICS 2021, LNCS 12972, pp. 240–260, 2021.
https://doi.org/10.1007/978-3-030-88418-5_12

of a key escrow mechanism (e.g., [24]) and file I/O pattern profiling (e.g., [11, 12, 19, 21, 28, 30, 32, 33, 37]). However, ransomware still remains the most popular cyber threat in the real world.

Attackers are now developing and employing various attack vectors (called *stealthy* malware) to make ransomware much stealthier than before to evade the target system [35]. For instance, Windows operating systems (OSes) are shipped with highly-trusted administrative tools, such as Windows Management Instrumentation (WMI) and Microsoft PowerShell (PS), to assist system administrators in performing their daily routine tasks. However, stealthy ransomware is trying to evade detection and prevent file recovery by impersonating benign processes. For instance, variants of Ryuk ransomware use the WMI tool to delete Windows shadow files (using the command `wmic.exe shadowcopy delete`) to prevent system restoration. Similarly, Virlock ransomware leverages the Console Registry tool for Windows (*reg.exe*) to add registry keys and values in order to disable User Account Control (UAC) check to remain stealthy. Moreover, stealthy ransomware would attempt to hide malicious code through various packing techniques and unpack only into the memory called "living off the land" attack (fileless malware) to evade disk scanning tools. Fileless malware is a type of malicious software that secretly activate pre-installed tools and applications on victims' computer to execute an attack. Recent security reports reveal that fileless malware attack rates skyrocketed by almost 900% in 2020 compared to 2019 [36]. Antivirus products may not be effective anymore in detecting such ransomware as they search for malicious payload (i.e., *signature*) on disk, whereas fileless malware is memory-based and is loaded from the remote server using pre-install utilities. Finally, the ransomware can exploit vulnerabilities in benign programs to gain control. Because attackers have many opportunities to bypass systems' security and generate new malware variants, the detection approaches based on static or behavioral signatures cannot catch up with the ever-evolving stealthy ransomware [20].

To address the shortcomings identified above, we develop a novel ransomware detection system, called "Peeler" (**P**rofiling k**E**rn**E**l -**L**evel **E**vents to detect **R**ansomware), which relies on kernel-level provenance monitoring to capture contextual and behavioral characteristics of ransomware. Peeler intercepts kernel-level system events from the native layer of Windows OS and tracks ransomware activities from a ransomware binary execution on a victim's computer to the display of a *ransom note* on the victim's screen. To facilitate better defence against stealthy ransomware attacks on Windows OSes, Peeler profiles the contextual and behavioral characteristics of the executing processes. For instance, Hendler et al. [15,16] proposed deep learning approaches to detect malicious *PowerShell* commands. However, in practice, malicious commands could be launched not only by *PowerShell*, but also from Windows OS legitimate utilities, such as *vssadmin*, *wmic*, etc. We consider all malicious command sources rather than relying on malicious commands executed from a specific program. Applications perform various tasks using Windows native tools, and the context of operations depends on the commands/codes executed using these tools. Peeler determines the

context of the operations by extracting the command-line argument codes/scripts from the processes and builds a natural language processing (NLP)-based deep learning model to fingerprint applications' behavior for detection. Moreover, Peeler exploits some key behavioral characteristics of ransomware relating to relationships between certain kernel-level system events. For example, file I/O's *Read* and *Write* events are strongly correlated during ransomware execution because all the *Read* events are followed by the *Write* events in order to create an encrypted version of users' files. Such characteristics are exhibited in events from other system events as well, such as *Process* provider's *Start* and *Stop* and DLL *Image* provider's *Load* and *Unload* events, respectively. Consequently, the relationships between those events could be exploited via extracting features for machine learning models that could then perform detection. We develop a highly accurate ransomware detection system for Windows operating systems (OS)[1]. Our key contributions are summarized below:

- We develop a set of methodologies based on a comprehensive analysis of ransomware from binary execution on a victim's computer to the display of a *ransom note* at the victim's computer screen and design a fast and highly accurate ransomware detection system called Peeler.
- We present a novel transformer-based language model to fingerprint the contextual behavior of applications by leveraging Bidirectional Encoder Representations from Transformers (BERT) pre-trained model. To the best of our knowledge, our work is the first to apply transformer-based deep learning model for ransomware detection.
- We evaluate the performance of Peeler. Peeler achieves 99.52% detection accuracy with a false positive rate of only 0.58% against 67 ransomware families – the largest dataset so far in terms of its diversity. To show the robustness against unseen ransomware attacks, we also use Peeler (without new training) against 70 samples from 26 new and unseen ransomware families, including crypto and screen-locker. Peeler still achieves more than 95% detection accuracy in detecting new and unseen ransomware samples.

2 Related Work

We categorize the literature regarding ransomware detection into three groups: 1) crypto ransomware detection techniques that are mainly based on specific behavioral indicators (e.g., file I/O event patterns), 2) machine learning-based approaches that build models by leveraging system behavior feature, and 3) decoy-based approaches that deploy decoy files and monitor if ransomware samples tamper with the decoy files.

Crypto Ransomware Detection. There were several proposals to monitor file I/O request patterns of applications to detect crypto ransomware.

[1] Windows OS is becoming the most attractive targets for ransomware writers, i.e., 87% of the existing ransomware were developed to target Windows [10].

Kharraz et al. [21] studied crypto ransomware families' file I/O request patterns and presented a dynamic analysis-based ransomware detection system called UNVEIL. UNVEIL detected 13,647 ransomware samples from a dataset of 148,223 general malware samples. Kharraz et al. [22] proposed another ransomware detection system using file I/O patterns, achieving a 100% detection rate with 0.8% false positive on 677 samples from 29 ransomware families. Scaife et al. [32] also presented a system called CryptoDrop that detects ransomware based on suspicious file activities, e.g., tampering with a large number of file accesses within a time interval. According to the experimental results, the number of lost files is ten on average. Moratto et al. [30] proposed a ransomware detection algorithm with a copy of the network traffic without impacting a user's activities. Their proposed system achieved a 100% detection rate with 19 different ransomware families after the loss of ten files.

Machine Learning Based Ransomware Detection. RWGuard [27] is a machine learning-based crypto ransomware detection system. It achieved a 0.1% false positive rate incurring a 1.9% CPU overhead with 14 crypto ransomware families. RWGuard leverages the features about processes' I/O requests and changes performed on files because RWGuard was mainly designed to detect crypto ransomware only. Sgandurra et al. [33] proposed EldeRan, another machine learning approach that builds a model using system activities such as API invocations, registry event, and file operations that are performed by applications. EldeRan achieved a 93.3% detection rate with a 1.6% false alarm rate with 582 samples from 11 ransomware families. Hirano et al. [17] and Alrimy [11] proposed behavior-based machine learning models for ransomware detection. Hirano et al. selected five-dimensional features extracted from ransomware and benign applications' I/O log files. Nieuwenhuizen [31] proposed another behavioral-based machine learning model using a feature set that quantifies the behavioral traits for ransomware's malicious activities.

Decoy Files Based Ransomware Detection. Decoy techniques [12,14,27] have also been frequently proposed to detect ransomware attacks. For example, Gomez et al. [14] developed a tool called R-Locker using honey files to trap the ransomware. When file operations are performed on honey files by a process, the process is detected and completely blocked because benign processes do not perform any file operations on honey files. However, if decoy files are generated, which look different from genuine user files, sophisticated ransomware samples ignore decoy files [27]. Moreover, it is also unclear how those solutions would detect some ransomware families (e.g., Petya) that affect predefined system files only.

3 Key Characteristics to Detect Ransomware

Typically, a ransomware attack consists of three stages: perform stealth operations to remain undetected, launch the actual attack, and display *ransom note* after a successful attack. Peeler exploits these differences in system behavioral characteristics between ransomware and benign applications.

3.1 Application Contextual Behavior

Application Process Tree. Applications can spawn one or more processes if it is needed to perform target tasks. If a process in an application creates another process, then the creator process is called *parent* process, and the created process is called *child* process. We observe that both ransomware and benign applications spawn child processes in a specific pattern depending upon the application logic. For example, typical ransomware should create several processes to perform malicious tasks (e.g., connecting to C&C servers, modifying the Windows registry, hiding files/extensions, encrypt user files) in parallel, resulting in a specific pattern of process creation called application process tree. The sequence of process creation contains rich contextual information highlighting the context of operations performed. These actions containing malicious command-line argument command codes and embedded or referenced PS scripts play an important role in determining the application's context. Figure 1 shows a snapshot of the application process tree of VirLock ransomware during its execution. VirLock is self-reproducing ransomware that not only locks a victim's screen but also infects her files. Both behaviors – self-reproducing and infecting files – were observed in the application process tree of VirLock. We can see *locker.exe* is replicated at level 3 of the tree.

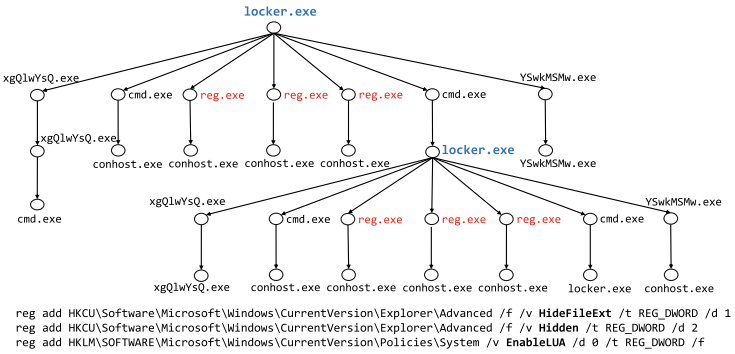

Fig. 1. Application process tree of VirLock.

To infect a victim's files, VirLock performs stealthy malicious activities to deceive victims. For instance, while creating files in the victim's computer, VirLock modifies the registry in the following ways: 1) disable Windows OS's User Account Control (UAC), which is a feature that was designed to prevent unauthorized changes in desktop computers; 2) hide all files that are created on the victim's desktop; and 3) hide all created file extensions. As shown in Fig. 1, three child processes (*reg.exe* in red color) perform the actual registry modifications. The corresponding command line execution is shown at the bottom of Fig. 1. In contrast, most benign applications create a considerably fewer number of spawned processes than stealthy ransomware. Figure 2 shows the application

process trees of six benign applications (Chrome, Adobe Acrobat Reader, MS Visual Studio 2019, MS Office 365 ProPlus, Spotify, and MS Outlook).

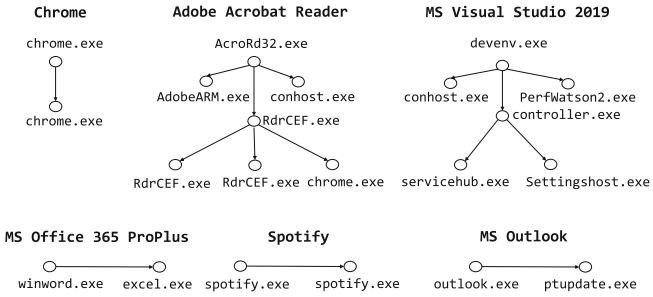

Fig. 2. Application process trees of benign applications.

Our observations on the behaviors of ransomware families show that more than 60% of samples spawn multiple processes. For instance, the VirLock ransomware sample spawns 44 processes on average. On the other hand, our observations on more than 50 most popular benign Microsoft applications show that they spawn 16 processes on average. We surmise that the processes created from both ransomware and benign apps can provide sufficient contextual information which could be leveraged to infer the intent of the underlying application.

Malicious Commands. To maximize the impact of the encryption, ransomware performs malicious activities with the following three goals: stealthiness, attack launch, payment guidance. 1) *Stealthiness:* ransomware tries to remain undetected by anti-malware services or Windows OS defenders running on the victim's computer. For example, they may disable runtime monitoring, archive scanning, automatic startup repair. Figure 1 shows the set of malicious commands executed by the Virlock ransomware to achieve stealthiness. Moreover, they may delete ransomware executable from the disk, stop all anti-malware services, or turn off User Account Control (UAC). 2) *Attack:* ransomware deletes the shadow copies and the system's backup/restore data automatically created by Windows OS. For example, `vssadmin` is a legitimate Windows OS utility that controls volume shadow copies of user files on a given computer. These shadow copies are regularly used as a recovery point, and additionally, they can be leveraged to re-establish or return the file to a previous state if they are destroyed or lost due to some reasons. Adversary exploits `Vssadmin` utility by executing the command `vssadmin.exe delete shadows /all /quiet`, to delete Windows OS shadow copies, making it impossible to restore the system back to its previous state. Note that an adversary can also use other legitimate tools pre-installed on Windows OS to delete shadow copies such as `PowerShell`, `wmic`, etc. The ransomware can leverage Windows OS program `net.exe` commands to stop or bypass detection by several popular antivirus software, in

addition to defeating Windows OS Defender, e.g., `net.exe stop avpsus /y` stops Windows OS process Kaspersky Seamless Update Service, which is used by Thanos ransomware family. Moreover, ransomware sometimes tries to kill the processes related to specific programs, such as SQL server, to initiate the encryption of the user files on which these programs were operating. After encrypting user files, ransomware shows a ransom note and payment guidelines. 3) *Post-attack guidance on ransom payment* finally, a ransom note is displayed along with a read-me document to help the victim pay the ransom and restore the system files. The ransomware writer typically adds a registry key to the autorun path to show the ransom note window to achieve this. By intercepting such malicious commands at early stages, ransomware could be detected with no file losses.

3.2 Application Behavioral Characteristics

We analyzed the system events generated by ransomware samples and observed that there exist strong correlations between some events for the operations of ransomware. For example, all files read must be written (encrypted), which naturally shows a correlation between `Read` and `Write` events. Furthermore, similar correlations are exhibited among events collected from different providers such as `File`, `Process`, `Image`, and `Thread` because ransomware samples generate a large file `Read` and `Write` events to perform malicious tasks. Such relationships between certain events can be quantified by using the correlation coefficients of the events (see Table 1).

Table 1. Correlation coefficients for some events.

Events pair	Ransomware	Benign applications
(File Read, File Write)	0.9433	0.3500
(Process End, Image Unload)	0.9451	0.7174
(Process Start, Image Load)	0.9476	0.7397
(Thread Start, Thread End)	0.9560	0.6585

For example, a crypto ransomware sample generates `Read` and `Write` events regularly. As presented in Table 1, there exists a strong correlation between the number of `Read` events and the number of `Write` events. Such a correlation relationship may not appear in benign applications' `Read` and `Write` requests. Similarly, during ransomware execution, we observe the correlation between the number of `Start` processes and the number of image `Load` events, the correlation between the number of `End` processes and the number of image `Unload` events, and the correlation between the number of `Start` threads and the number of `End` thread events. These correlation coefficients are computed from the analysis performed on 206 ransomware samples and 50 most popular benign

applications. Figure 3 shows correlation among three pairs of events (`Read` and `Write`, `Start` and `Load`, `End` and `Unload`). We clearly observe strong correlations for ransomware compared to benign applications. Therefore, Peeler uses those correlations to detect ransomware.

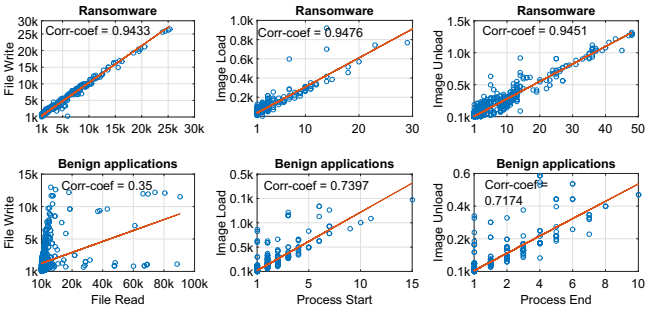

Fig. 3. Correlations among events for ransomware and benign applications.

4 System Design

4.1 Overview

Peeler has two main ransomware detection modules: 1) malicious command detector and 2) machine learning-based classifier. Figure 4 illustrates the overall design of Peeler. Peeler monitors system events continuously to detect ransomware attacks in real-time and uses them to perform ransomware detection. The malicious command detector uses predefined rules to check whether malicious commands are executed by processes in which the execution of those commands is mainly observed in ransomware activities. Machine learning-based classifier module captures contextual and behavioral features using two machine learning models. Peeler profiles the contextual information of created processes by extracting features from process command-line arguments (see Table 2). It then transforms the extracted features into contextual embedding to build an NLP-based deep learning model called BERT [13]. On the other hand, Peeler constantly monitors and extracts behavioral features the following providers: `Process`, `Image`, `File`, and `Thread` and builds another machine learning model based on support vector machine (SVM). For detection, the scores from both classification models are fused as an ensemble approach, and then detection is performed.

4.2 System Events Monitor

Peeler provides a module called *system events monitor* which relies on Event Tracing for Windows[2] (ETW) framework. ETW is a built-in, general-purpose

[2] ETW was first introduced in Windows 2000 and is now built-in to all Windows OS versions.

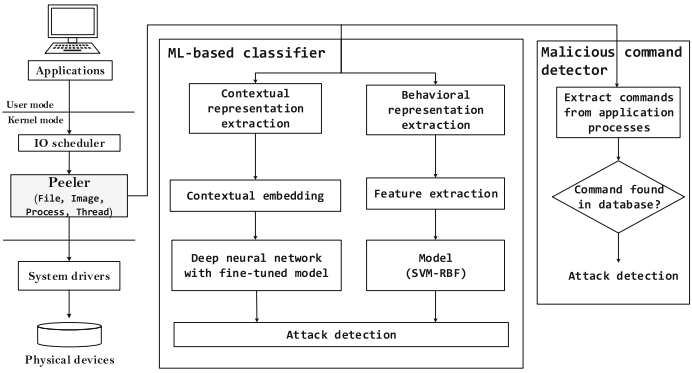

Fig. 4. Overview of Peeler.

logging and diagnostic framework in Windows OS. It is *efficient* (high speed and low overhead), *flexible* (consume events in real-time or log to a file), and provide greater *visibility* into the system such that it allows to register to more than 1,100 subsystem providers [1] to receive and consume events. ETW framework is available for usage in both APIs and command-line tools and applications. The usage of ETW-based command-line tools and applications are more common compared to ETW APIs (e.g.,`TraceEvent`, `System.Diagnostic.Eventing`, `Event Tracing`) to view system events [26].

We designed a module called *system events monitor* based on ETW. Our module is lightweight because it directly interacts with the native layer and performs filtering of the system events. In terms of visibility, Peeler extracts events from the following providers: `Process, Image, File, Thread`. The data obtained by Peeler's system events monitor are in the form of a continuous sequence of events t_i. An event is represented as: $t_i =< PID, TID, Prov., EType, E_{timestamp}, E_{attrs} >$, where PID is a process identifier, TID is a thread identifier corresponding to the process PID. Prov. is provider name, EType is event name,$E_{timestamp}$ is the time of event occurrence, and E_{attrs} is a set of attributes of the event E_{name}. To implement the system events monitor in Peeler, we used an open-source project *krabsetw* [3], which is a C++ library that simplifies interactions with ETW. We modified *krabsetw* both at API and native layer to collect the events only needed for Peeler.

Table 2. Providers and events used in Peeler.

Provider	Event	Event schema	
		Common attributes	Provider-specific event attributes
Process	Start, End	PID, TID, Prov., Event, Timestamp	SessionId, ParentId ImageFileName, CommandLine

4.3 Malicious Commands Detector

Peeler uses a component called *malicious command detecter* to filter suspicious activities conducted by ransomware using the database for malicious commands which are needed to perform ransomware's activities. Peeler can collect malicious commands from the `Process`'s `Command line` argument (see Table 2). Typically, the `commandLine` attribute of the `Process` contains the actual commands. For example, Windows OS utility `net.exe` can start, stop, pause or restart any service using the command `net.exe stop ServiceName` launched via convenient script/batch file or command prompt. If an adversary leverages such commands to stop several anti-malware services, Peeler can detect the process using predefined rules. Similarly, utility *taskkill.exe* ends one or more tasks or processes. Typically, ransomware specify the image name of the process to be terminated (e.g., *taskkill.exe /IM ImageName*). The `sc.exe` utility modifies the value of a service's entries in the registry and service control manager database. Ransomware may attempt to disable several defending services on the victim's machine before starting encryption. Peeler uses rules based on utility names and actions performed in order to detect ransomware infection.

4.4 Machine Learning-Based Classifier

Peeler uses two different machine learning-based (ML-based) classifiers with contextual features and system events-based behavioral features to detect ransomware samples. The input to the algorithm is a set of events accumulated over W seconds window. In our current implementation, we empirically set $W = 5$ to optimize the speed-accuracy tradeoff. Features are extracted to build two machine learning models. The built machine learning models are then used to detect ransomware attacks.

Building the First Classifier with the Applications' Processes. As discussed in Sect. 3.1, we observed that several ransomware samples, as well as benign application samples, spawn many child processes (depending upon the application logic) to perform different tasks. However, to determine whether or not a given process's operations are malicious or not may depend on its context of operation because, like benign applications, stealthy ransomware also uses legitimate tools on Windows OS to perform various operations. To that end, Peeler profiles an application's contextual behavior by exploiting the processes creation pattern (including the commands executed during the processes runtime).

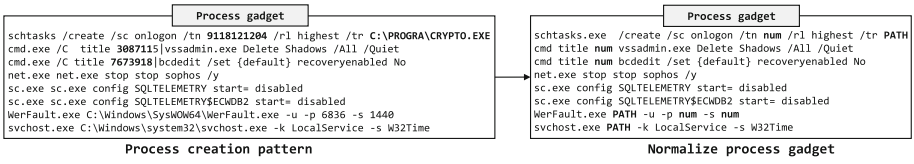

Fig. 5. Process gadget representation and normalization.

A set of semantically related processes is called *process gadget*. Process gadgets partially depict the application's contextual behavior. For instance, Fig. 5 shows the process creation pattern containing the used tools information and the command executed. To view it contextually, ransomware first scheduled a high-priority task using the *schtasks* tool to execute a malicious file. Then it deletes Windows shadow files using *Vssadmin* and *cmd* tools. Then the recovery and telemetry information is disabled using *sc.exe* and *net.exe* tools, and so on. The process gadget in Fig. 5 provides sufficient context about the intend of the executing application, which is exploited by Peeler to detect ransomware. In addition, Peeler normalizes the process gadgets by replacing numerical values with literal constants such as 'num' and absolute paths with 'PATH' constant, as shown in Fig. 5 (right). Further, we observed that in some cases, PowerShell use encoded (with Base64 encoding scheme) malicious PS commands. In such cases, Peeler first decodes (from Base64 encoding) the commands and then uses the obtained text of commands in process gadgets.

To that end, we propose to use an NLP-based deep neural network called BERT [13] that takes the process gadgets and constructs a new deep learning language model via fine-tuning. BERT [13] is pretrained by considering two main objectives: 1) Masked language modeling (MLM): the model randomly masks 15% of the words in the input, then runs the entire masked sentence through the model and has to predict the masked words. This approach is different from traditional recurrent neural networks (RNNs) that usually see the words sequentially, i.e., one after the other, or from autoregressive models like GPT, which internally mask the future tokens. BERT's main advantage is that it allows the model to learn a bidirectional representation of the sentence – providing context to the learning stage. 2) Next sentence prediction (NSP): the models concatenate two masked sentences as inputs during pretraining. The goal here is to predict if the two sentences were following each other or not.

In the fine-tuning step, the model weights are updated using our labeled dataset, which is new to the pretrained BERTbase model. BERT takes a sequence of tokens in a maximum length of 512 as input and produces a sequential representation in a 768-dimensional vector as output [13]. The network layers need to be updated to train the deep learning model on a new classification task, i.e., to identify benign/ransomware applications. The output layer is then trained to display the result intended for the fine-tuned deep learning model. We train a classifier with different layers of 768 dimensions on top of the pre-trained BERTbase transformer model to minimize task-specific parameters through the fine-tuning step.

Building the Second Classifier with the System Event Features. As discussed in Sect. 3.2, Peeler leverages four providers' (`File`, `Process`, `Image`, and `Thread`) events exhibiting casualties. In total, the following four pairs of events are used: (`Read`, `Write`), (`Start`, `Load`), (`End`, `Unload`), and (`Start`, `End`). To capture these casualties simply, Peeler extracts frequency features and trains an SVM model based on feature set FV_{SVM} for classification. We selected SVM

with RBF kernel because it is lightweight and produces the best accuracy results with FV_{SVM}.

Features: # of process start, # of process end, # of DLL image loads, # of DLL image unloads, # of file reads, # of file writes, # of threads start, # of thread end

Attack Detection. Peeler uses two classification models (fine-tuned BERT and SVM-RBF) and finally decides the classification outcome by fusing their scores. Peeler extracts process command-line arguments from the stream of incoming system events, transforms them into their normalized representations, and uses them as input to the fine-tuned deep learning model. The constructed model network outputs either "1" (i.e., ransomware) or "0" (i.e., benign). If a given file event gadget is from ransomware, it will extract the process identifier and flag the process by raising an alert to the system administrator. SVM-RBF is trained with FV_{SVM}. The scores from the two models are fused by taking their average for detection.

5 Dataset Collection

5.1 Ransomware

We collected 28,034 ransomware samples from VirusTotal [9], MalwareBazaar [6], malware repository [4], malwares [5], and other online communities. However, we excluded many malware samples from our final dataset for experiments. First, we found that many samples were not actual ransomware samples, although they were classified as ransomware by some vendors in VirusTotal. Therefore we discarded such samples. This finding is consistent with the observation in the previous work [32]. Second, ransomware often needs to interact with command-and-control (C&C) servers to perform their malicious activities. However, several ransomware samples did not often work appropriately because their corresponding C&C servers were inactive. Also, some sophisticated malware samples can detect the analysis environment and remain inactive to evade detection [29]. More importantly, samples from a few ransomware families we observed were significantly larger compared to other families. For example, we found more than 20,000 ransomware samples from Virlock family including *Virlock Gen.1*, *VirLock Gen.4*, and *VirLock Gen.8* variants. Therefore, we kept limited samples from the Virlock family and discarded other samples. Finally, we collected 292 active samples from 67 ransomware families that perform their activities correctly. We used 206 ransomware samples from 43 ransomware families (see Appendix A) in the first set of experiments (in Sect. 6.1). The remaining 24 ransomware families with 70 samples were collected at a later stage of data collection and used to evaluate Peeler on new and unseen ransomware samples (in Sect. 6.3).

User Environment and Ground Truth (labeled) Dataset. We used VirtualBox 6.1 [8] to create and manage the computing environment locally for experiments. Rather than using artificially generated data, we used a real user's data running on the Windows 10 64Bit operating system (a copied version of real user data) to set up a benign user's environment realistically. Multimedia files (e.g., `bmp`, `jpeg`, `png`, and `mp4`), Microsoft office documents (e.g., `docx`, `xlsx` and `pptx` files), and other important files (e.g., `cpp`, `py`, `pdf` and `wav` files) were copied to various directories in different locations. We note that those files are typically most attractive targets for ransomware.

Each ransomware sample was executed and then manually labeled by each family type. We ran each ransomware sample for ten minutes or until all user files were encrypted. It took more than 90 days to run all samples and collect data. We only considered those samples that encrypted user files or locked desktop screens. If no files were modified, we excluded them from our dataset. We also obtained labeled ransomware samples from two well-known malware repositories [4,6].

Diversity in Our Dataset. Table 6 presents a list of ransomware families that are used in our evaluation. To the best of our knowledge, this is the most comprehensive dataset containing diverse ransomware families. According to previous work [31,32], the use of diverse families is more important than the number of ransomware samples from a few families for evaluating the performance of ransomware detectors. For instance, building a model on 1,000 Locky (and its variants) ransomware samples should prove no more useful than building a model on just one Locky sample [31]. Scaife et al. [32] confirmed that due to the homogeneous nature of file I/O behavior among samples within each family, a small number of representative samples in each family are sufficient to evaluate the detection performance. The core behavioral traits shown by crypto ransomware in encrypting data remain almost identical from one variant to another in a family. Since our study covered more than eight times the number of families from previous study [23], and more than two times the number of families covered in studies [21,32] and there was not much diversity within families, there was little need to collect additional samples.

5.2 Benign Applications

We also collected the dataset for popularly used applications. In addition to popularly used applications, we also considered several benign applications that could resemble ransomware in certain behavioral aspects. The reason is to investigate false-positive rates when benign applications potentially resemble ransomware. We divide the benign dataset into three main categories targeting various types of ransomware. The first category applications perform encryption or compression operations that would generate file I/O patterns similar to crypto ransomware that could result in false positives. The second category applications spawn many child processes. The third category applications are popularly used applications on Windows PCs. We collected user's system usage data under

normal conditions while interacting with those applications. A user runs many different applications at the same time. For example, the user reads a document using Adobe Acrobat Reader, switched to the internet browser to view online reviews about a product, and then used Adobe Acrobat Reader again. The list of benign applications is shown in Appendix A.

6 Evaluation

We demonstrate Peeler's performance in detection accuracy, identification of legitimate tools abused by an attacker, and robustness against unseen ransomware.

For evaluation, we used the dataset described in Sect. 5. Our dataset consists of 177,236,131 system I/O events containing both ransomware samples and benign applications. Table 3 shows a detailed breakdown of our dataset. For training, 60% of both ransomware and benign applications are selected randomly. The remaining ransomware and benign samples are divided into validation (10%) and test (30%) datasets.

Table 3. Dataset statistics.

Dataset	Ransomware	Benign	Total
# system events	66,099,039	111,137,092	177,236,131
# processes	3,929	3,924	7,853
# process gadgets	1,309	1,308	2617

For training the SVM-RBF model, we used default settings to train the model. For fine-tuning the BERT model with our dataset, we used BERTbase [7] tokenizer and pretrained model. The number of commands in each process gadget is set to three because, with just three commands, BERT is able to capture the contextual information. A total 2,617 number of process gadgets are extracted from 7,853 processes from both ransomware and benign applications. The model consists of 12 layers, 12 attention heads, 768-dimensional vocabulary, and 110M parameters. The batch size and learning rate were set to 16 and 0.00002, respectively. The number of epochs was set to 10. The trained model is then used to evaluate Peeler.

6.1 Detection Accuracy

Table 4 shows the summary of Peeler's detection accuracy. Overall, Peeler achieved 99.52% accuracy with a false positive rate of only 0.58%. Similarly, Peeler achieved over 99% in both precision and F1 score.

Table 4. Peeler's detection accuracy.

Acc. (%)	TPR (%)	FPR (%)	FNR (%)	Prec. (%)	Rec. (%)	F1 (%)
99.52	99.63	0.58	0.37	99.41	99.63	99.52

False Positive Analysis. Minimizing false positives is essential to develop practically useful malware detectors because excessive false positives can annoy users and undermine the system's effectiveness. We evaluate Peeler's performance against three different types of benign application scenarios (see Table 5).

Table 5. Peeler's false positive analysis.

Scenario	TNR (%)	FPR (%)	FNR (%)
Crypto-like benign apps	98.27	1.72	0.96
Locker-like benign apps	99.5	0.31	0.5
Commonly used benign apps	99.78	0.21	0.87
All ransomware	99.42	0.58	0.37

Crypto Ransomware-Like Benign Applications. For behavior-based crypto ransomware detection solutions, a significant challenge is not to detect benign applications having compression or encryption capabilities because their system behaviors might be similar to crypto ransomware.

We deeply investigated 11 different applications using compression or encryption operations on a large number of files like crypto ransomware (see Appendix A). We observed that event sequences of some benign applications such as ZipExtractor and BreeZip are similar to those of typical crypto ransomware, but they do not restrict access to files via encryption, unlike crypto ransomware. Table 5 shows that Peeler correctly detects 98.27% with a false positive rate of 1.72% even against crypto ransomware-like benign applications.

Benign Applications Spawning Many Processes. Certain benign apps spawn many child processes. Therefore, we examine how Peeler's performance can be degraded with benign apps having such behaviors. For this analysis, we investigated 34 most popular applications from Microsoft's Windows OS Store (https://www.microsoft.com/en-us/store/apps/windows) (see Appendix A) and selected 18 applications showing such behaviors. We observe that Pycharm, Visual Studio, and Chrome spawned 140, 46, and 42 child processes, respectively. We examined the results of Peeler with those 18 applications. Table 5 shows that Peeler correctly detected 99.5% with a false positive rate of 0.31% even though these benign applications spawned many processes.

Popularly Used Benign Applications. We also evaluated Peeler's performance with commonly used benign applications such as Microsoft Office, Adobe

Acrobat Reader, email client, and instant messengers, as presented in Sect. 5.2. We show that Peeler correctly detects all benign activities performed by a user achieving a detection rate of 99.78%. The false positive and true negative rates under normal system usage are 0.21% and 0.87%, respectively. Note that the overall detection accuracy in all three scenarios is above 99%.

6.2 Effectiveness in Detecting Abused Tools/utilities

Some sophisticated ransomware samples can exploit legitimate tools/utilities, called Living-Off-The-Land techniques, to perform malicious tasks. It is therefore essential to identify such tools/utilities and the malicious tasks performed by them. To show the effectiveness of our approach, we compare Peeler with PROVDETECTOR [35] in terms of detecting the number of legitimate tools/utilities abused by ransomware. It is an anomaly-based malware detection approach that models system behavior using graphs. Total 23 legitimate tools or applications are detected by PROVDETECTOR that are abused by malware. However, Peeler detected 65 tools and applications (almost three times more than that of PROVDETECTOR) that can be abused by ransomware in performing malicious tasks. The list of abused legitimate tools and applications by ransomware is given below:

raserver.exe, SystemSettings.exe, svchost.exe, SpeechRuntime.exe, SystemSettingsBroker.exe, SearchProtocolHost.exe, SearchFilterHost.exe, cmd.exe, reg.exe, cscript.exe, backgroundTaskHost.exe, RuntimeBroker.exe, HxTsr.exe, SIHClient.exe, WmiPrvSE.exe, WindowsInternal.ComposableShell.Experiences.TextInput.InputApp.exe, provtool.exe, taskhostw.exe, rundll32.exe, DiskSnapshot.exe, cleanmgr.exe, Defrag.exe, CompatTelRunner.exe, sdiagnhost.exe, TiWorker.exe, dllhost.exe, ShellExperienceHost.exe, SearchUI.exe, mobsync.exe, MicrosoftEdgeCP.exe, ngentask.exe, ngen.exe, LogonUI.exe, makecab.exe, taskkill.exe, ruby.exe, OpenWith.exe, notepad.exe, PING.EXE, mshta.exe, consent.exe, vssadmin.exe, WMIC.exe, wscript.exe, Windows.WARP.JITService.exe, forfiles.exe, bcdedit.exe, netsh.exe, Microsoft.Photos.exe, net.exe, net1.exe, schtasks.exe, mode.com, WerFault.exe, wermgr.exe, timeout.exe, drpbx.exe, SearchIndexer.exe, MusNotificationUx.exe, icacls.exe, CsWEBftw.exe, powershell.exe, unsecapp.exe, chcp.com, attrib.exe

6.3 Robustness Against Unseen Families

To test Peeler against unseen ransomware families, we additionally collected new and *unseen* ransomware samples after three months from the first experiments and monitored online repositories for new or unseen ransomware samples. A total of 70 samples from more than 24 distinct unseen or new ransomware families are tested. We used the previously constructed Peeler without retraining. All samples tested in this experiment are manually verified from VirusTotal and other online malware repositories to confirm their family and type. Peeler correctly detected

67 samples from a total of 70 new and unseen ransomware samples achieving more than 95% detection rate. The ransomware families and the number of corresponding samples tested are given in Appendix A.

7 Conclusion

We propose a new efficient tool called Peeler to detect ransomware attacks using their system behaviors. Peeler is built on both rule-based detection and machine learning models to improve detection accuracy. To show the effectiveness of Peeler, we evaluate its performance with 43 ransomware families containing crypto ransomware and screen-locker ransomware. In the experiments, Peeler achieved a 99.52% F1 score with a false positive rate of only 0.58%. Further, we showed that Peeler is highly effective in detecting unseen or new ransomware, achieving more than 95% detection accuracy on 24 unseen ransomware families.

A Dataset

A.1 Ransomware families

We provide a comprehensive list of both ransomware families and benign applications used to evaluate Peeler. Table 6 presents two sets of ransomware families used in Sect. 6.1 and Sect. 6.3, respectively.

Table 6. Ransomware families and samples.

No.	Family	Samples	No.	Family	Samples	No.	Family	Samples	No.	Family	Samples
1	Cerber	33	12	Petya	1	23	Sodinokibi	14	34	Satana	1
2	GoldenEye	12	13	Shade	1	24	Sage	5	35	Syrk	1
3	Locky	5	14	TeslaCrypt	1	25	Dharma	3	36	ucyLocker	1
4	dotExe	3	15	Unlock92	1	26	Troldesh	1	37	Vipasana	1
5	WannaCry	3	16	Xorist	2	27	Da Vinci Code	1	38	Malevich	1
6	Shield	1	17	Jigsaw	1	28	Cryptowire	1	39	Adobe	1
7	District	1	18	Virlock.Gen.5	83	29	Gandcrab	1	40	LockScreen.AGU	12
8	GlobeImposter	1	19	Alphabet	2	30	Hexadecimal	1	41	EgyptianGhosts	1
9	InfinityCrypt	1	20	Lockey-Pay	1	31	IS (Ordinpt)	1	42	Blue-Howl	1
10	Keypass	1	21	ShellLocker	1	32	Lockcrypt	1	43	DerialLock	1
11	Pack14	1	22	Trojan.Ransom	1	33	PocrimCrypt	1		–	
Dataset for experiments in Section 6.3.											
44	Ryuk	6	50	Zeppelin	6	56	Ranzy	4	62	Netwalker	2
45	Core	3	51	Fox	3	57	Crpren	1	63	MedusaLocker	1
46	Balaclava	5	52	Crylock	7	58	Matrix	4	64	DarkSide	4
47	RagnarLocker	2	53	HiddenTear	2	59	Mespinoza	5	65	Thanos	3
48	Vaggen	3	54	Mountlocker	2	60	Nemty	2	66	Phobos	1
49	Jsworm	1	55	Winlock	1	61	Maze	1	67	Unknown	1
Total samples: 292, crypto = 188, screen-locker = 104											

A.2 Benign applications

In this section, we present benign applications that potentially show ransomware-like behaviors that are used in the evaluation of Peeler: 1) benign encryption, compression, and shredder applications (see Table 7); 2) benign application spawning multiple processes; and 3) benign applications that are most popularly used on Windows PC (see Table 8).

Table 7. Ransomware-like benign applications.

Tool	Application	Operation	Version
Compression	7-zip	Compression	19.00
	7-zip	Decompression	
	Winzip	Compression	24
	Winzip	Decompression	
	Winrar	Compression	5.80
	Winrar	Decompression	
	BreeZip	Compression	–
	BreeZip	Decompression	
	Alzip	Compression	11.04
	Alzip	Decompression	
	PeazipWinrar	Compression	7.1.1
	PeazipWinrar	Decompression	
Encryption	AESCrypt	Encryption	310.00
	AESCrypt	Decryption	
	AxCrypt	Encryption	–
	AxCrypt	Decryption	
Shredder	Eraser	Delete	6.2.0.2986
	Ccleaner	Delete	–
	Windows Delete	Delete	–

Table 8. Benign applications spawning multiple processes.

Type	Application	Version	spawn processes?
Office	MS Word	16.0.11929.20436	×
	MS Powerpoint	16.0.11929.20436	×
	MS Excel	16.0.11929.20436	×
	MS Outlook	16.0.11929.20436	✓
	Trio Office: Word, Slide, Spreadsheet	–	✓
Development	Pycharm	11.0.3+12-b304.56 amd64	✓
	Matlab	R2019a	✓
	Visual Studio C++	2019 community version	✓
	Android Studio	191.6010548	✓
Tools	Adobe Acrobat Reader	20.006.20034	✓
	Adobe Photoshop Express	3.0.316	×
	PhotoScape	3.7	×
	Cool File Viewer	–	×
	PicArt Photo Studio	–	×
	Paint 3D	–	×
Cloud and Internet	Dropbox	–	×
	Googledrive	–	×
	Internet Explorer	11.1039.17763	✓
	Google Chrome	80.0.3987.132	✓
	Remote Desktop	–	×
Messenger	Telegram	1.9.7	×
	WhatApp	0.4.930	✓
	Skype	1.9.7	×
	Facebook Messenger	–	✓
Document	Wordpad	–	×
	Notepad	–	×
	OneNote	16001.12527.20128.0	×
Media player	VLC	3.0.8	×
	Netflix	6.95.602	×
	GOM Player	2.3.49.5312	×
Miscellaneous	Spotify	–	✓
	KeePass Password manager	1.38	×
	Discord	–	×
	Facebook	–	×

References

1. About Event Tracing. https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing
2. Global Ransomware Damage Costs Predicted To Reach $20 Billion (USD) By 2021. https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021/
3. Krabsetw. https://github.com/microsoft/krabsetw
4. A Live Malware Repository. https://github.com/ytisf/theZoo

5. Malware samples. https://github.com/fabrimagic72/malware-samples
6. MalwareBazaar. https://bazaar.abuse.ch/
7. Pretrained models. https://huggingface.co/transformers/pretrained_models.html
8. VirtualBox. https://www.virtualbox.org
9. VirusTotal. https://www.virustotal.com/
10. What systems have you seen infected by ransomware? https://www.statista.com/statistics/701020/major-operating-systems-targeted-by-ransomware/
11. Al-rimy, B.A.S., Maarof, M.A., Shaid, S.Z.M.: A 0-day aware crypto-ransomware early behavioral detection framework. In: International Conference of Reliable Information and Communication Technology, pp. 758–766 (2017)
12. Continella, A., et al.: Shieldfs: a self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 336–347 (2016)
13. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
14. Gómez-Hernández, J., Álvarez-González, L., García-Teodoro, P.: R-Locker: thwarting ransomware action through a honeyfile-based approach. Comput. Secur. **73**, 389–398 (2018)
15. Hendler, D., Kels, S., Rubin, A.: Detecting malicious powershell commands using deep neural networks. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pp. 187–197 (2018)
16. Hendler, D., Kels, S., Rubin, A.: Amsi-based detection of malicious powershell code using contextual embeddings. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, pp. 679–693 (2020)
17. Hirano, M., Kobayashi, R.: Machine learning based ransomware detection using storage access patterns obtained from live-forensic hypervisor. In: Sixth IEEE International Conference on Internet of Things: Systems, Management and Security (IOTSMS), pp. 1–6 (2019)
18. Homayoun, S., Dehghantanha, A., Ahmadzadeh, M., Hashemi, S., Khayami, R.: Know abnormal, find evil: frequent pattern mining for ransomware threat hunting and intelligence. IEEE transactions on emerging topics in computing (2017)
19. Huang, J., Xu, J., Xing, X., Liu, P., Qureshi, M.K.: Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 2231–2244 (2017)
20. Jin, B., Choi, J., Kim, H., Hong, J.B.: Fumvar: a practical framework for generating fully-working and unseen malware variants. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC) (2021)
21. Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., Kirda, E.: UNVEIL: a large-scale, automated approach to detecting ransomware. In: 25th USENIX Security Symposium (USENIX Security 16), pp. 757–772 (2016)
22. Kharraz, A., Kirda, E.: Redemption: real-time protection against ransomware at end-hosts. In: International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 98–119 (2017)
23. Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L., Kirda, E.: Cutting the gordian knot: a look under the hood of ransomware attacks. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 3–24 (2015)

24. Kolodenker, E., Koch, W., Stringhini, G., Egele, M.: Paybreak: defense against cryptographic ransomware. In: Proceedings ACM on Asia Conference on Computer and Communications Security, pp. 599–611 (2017)
25. Lab, E.M.: The State of Ransomware in the US. https://blog.emsisoft.com/en/34822/the-state-of-ransomware-in-the-us-report-and-statistics-2019/
26. Lelonek, B., Rogers, N.: Make ETW greate again. https://ruxcon.org.au/assets/2016/slides/ETW_16_RUXCON_NJR_no_notes.pdf
27. Mehnaz, S., Mudgerikar, A., Bertino, E.: RWGuard: a real-time detection system against cryptographic ransomware. In: International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 114–136 (2018)
28. Milajerdi, S.M., Eshete, B., Gjomemo, R., Venkatakrishnan, V.: Poirot: aligning attack behavior with kernel audit records for cyber threat hunting. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security, pp. 1795–1812 (2019)
29. Miramirkhani, N., Appini, M.P., Nikiforakis, N., Polychronakis, M.: Spotless sandboxes: evading malware analysis systems using wear-and-tear artifacts. In: IEEE Symposium on Security and Privacy (SP), pp. 1009–1024 (2017)
30. Morato, D., Berrueta, E., Magaña, E., Izal, M.: Ransomware early detection by the analysis of file sharing traffic. J. Network Comput. Appl. **124**, 14–32 (2018)
31. Nieuwenhuizen, D.: A behavioural-based approach to ransomware detection. Whitepaper, MWR Labs Whitepaper (2017)
32. Scaife, N., Carter, H., Traynor, P., Butler, K.R.: Cryptolock (and drop it): stopping ransomware attacks on user data. In: 36th IEEE International Conference on Distributed Computing Systems (ICDCS), pp. 303–312 (2016)
33. Sgandurra, D., Muñoz-González, L., Mohsen, R., Lupu, E.C.: Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. arXiv preprint arXiv:1609.03020 (2016)
34. Sivakorn, S., et al.: Countering malicious processes with process-dns association. In: Network and Distributed Systems Security (2019)
35. Wang, Q., et al.: You are what you do: Hunting stealthy malware via data provenance analysis. In: Symposium on Network and Distributed System Security (NDSS) (2020)
36. WatchGuard: Internet Security Report - Q4 2020. https://www.watchguard.com/wgrd-resource-center/security-report-q4-2020
37. Zhao, L., Mannan, M.: TEE-aided write protection against privileged data tampering. arXiv preprint arXiv:1905.10723 (2019)