Peeler: Profiling Kernel-Level Events to Detect Ransomware

***Abstract*—Ransomware is a growing threat that typically operates by either encrypting a victim's files or locking the desktop screen of a victim's computer until the victim pays a ransom. However, it is still challenging to detect timely such malware with existing traditional malware detection techniques. In this paper, we present a novel ransomware detection system, called "Peeler" (Profiling kErnEl -Level Events to detect Ransomware). Peeler deviates from the use of signatures for individual ransomware samples and relies on common and generic characteristics of ransomware depicted at the kernel-level. Analyzing diverse ransomware families at the kernel-level, we observed ransomware's inherent behavioral characteristics such as file I/O request patterns, process spawning, and causality relationships among kernel-level events. Based on those characteristics, we develop Peeler that continuously monitors kernel events of a target system and detects ransomware attacks on the system. Our experimental results show that Peeler achieves more than 99% detection rate with 0.58% false-positive rate against 43 distinct ransomware families, containing samples from both crypto and screen-locker types of ransomware. For crypto ransomware, Peeler detects them promptly after only one file is lost (within 115 milliseconds on average). Peeler utilizes around 4.9% of CPU time with only 9.8 MB memory under the normal workload condition. Our analysis demonstrates that Peeler can efficiently detect diverse malware families by monitoring their kernel-level events.**

I. Introduction

Ransomware is a class of malware that has significantly impacted enterprises and consumers. The goal of such type of malware is to obtain financial gain by holding a victim's system or resources as a hostage through encrypting the victim's files (called `crypto` ransomware) or locking the victim's desktop screen (termed as `screen-locker` ransomware). Recent statistics of ransomware shows that in the first three quarters of 2019, 151.9 million ransomware attacks were launched targeting enterprises and consumers [1], [2]. The average payment to release files to the victims spiked to $84,116 in the last quarter of 2019, more than double what it was in the previous quarter [3]. 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 [4]. As such, ransomware represents one of the most visible threats to enterprises as well as users. Therefore, a large number of proposals have recently been proposed to fight against ransomware as follows: machine learning models (e.g., [5], [6], [7], [8], [9], [10], [11]), use of decoy files (e.g., [12], [13], [9]), and file I/O pattern profiling (e.g., [14], [15], [16], [17], [18], [19], [11], [20], [9], [21]).

However, these techniques have at least one of the following limitations: 1) late detection after several files have already been encrypted, or the computer has been locked [15], [12], [22], [16], [11]. For instance, REDFISH [14] detects the ransomware activity when ten files are lost, and the detection time took around 20 seconds. Similarly, CryptoDrop [16] detects ransomware when ten files are lost. RWGuard [12] took 8.87 seconds on average to detect all malicious processes spawned by ransomware. 2) Most approaches rely on either rule-based detection methods (e.g., [15], [16]) or machine learning models (e.g., [12], [15]). Since each approach has its own advantages and disadvantages, we need to consider both approaches to build a practical ransomware detection system. 3) Most approaches mainly focus on crypto ransomware detection [12], [14], [16], [22] alone, while there are only a few approaches (e.g., [15]) that explicitly consider screen-locker ransomware. However, we need to consider both types of ransomware simultaneously because any type of ransomware attack can be launched in the real-world.

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 monitors kernel-level events and identifies suspicious event patterns generated by ransomware samples. Unlike existing approaches that rely on either rule-based detectors or machine learning models, Peeler is based on a hybrid approach that leverages both approaches to improve the detection time and detection accuracy at the same time. To achieve this goal, we intensively analyzed the system behaviors of both crypto and screen-locker ransomware and developed generic rules and features to detect both types of ransomware accurately. Consequently, unlike previous studies [15], [16] using *high-level* file I/O activities that should be translated from *kernel-level* file I/O events, we just used *kernel-level* file I/O events directly to distinguish crypto ransomware from benign applications. The suspicious file I/O event patterns are represented in regular expressions to detect crypto ransomware promptly. Besides, unlike existing approaches that rely on raw frequency features of several system-level events without analyzing their relationships, we analyzed ransomware samples' process properties and causality relationships among various kernel-level events generated during their execution to extract key features to build

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.23xxx
www.ndss-symposium.org

a machine learning model.

Peeler exploits the following three distinguishable characteristics between ransomware and benign applications: 1) In most crypto ransomware samples, we can find specific file I/O access patterns that are likely to appear to *overwrite* or *delete* user files. We observe that most ransomware families follow one of the four file I/O access patterns that could be exploited for detection. 2) Some ransomware samples spawn a large number of descendent processes while encrypting user files or locking desktop screens. Peeler captures such phenomena to detect ransomware attacks. 3) There are strong causal relationships between certain 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 providers as well, such as *Process* provider's *Start* and *Stop* and DLL *Image* provider's *Load* and *Unload* events, respectively. In consequence, the relationships between those events could be exploited via extracting features for machine learning models that could then perform detection. Peeler exploits the above mentioned characteristics by investigating a diverse set of ransomware samples. Using the key characteristics, we develop a fast and highly accurate ransomware detection system for Windows operating systems (OS)[1].

Our key contributions are summarized below:

- We analyze key characteristics to distinguish the system events of ransomware from those of benign applications and design a fast and highly accurate ransomware detection system, called Peeler, based on those characteristics. Peeler uses file I/O event patterns, and two classification models with 13 system behavior features to achieve both fast detection and high detection accuracy.
- We evaluate the detection accuracy of Peeler against 206 samples from 43 ransomware families containing both crypto ransomware and screen-locker ransomware. Overall, Peeler achieves 99.52% detection accuracy with a false positive rate of only 0.58%. Moreover, Peeler achieves 100% and 99.5% detection accuracy against crypto ransomware and screen-locker ransomware, respectively.
- We measure the detection time of Peeler against 102 samples from 34 crypto ransomware families and 104 samples from 9 different screen-locker ransomware families, respectively. Peeler took 115.3 milliseconds on average to detect crypto ransomware. Compared to the best existing crypto ransomware detection solution, Peeler is about five times faster. In addition, Peeler took 16.4 seconds on average to detect screen-locker ransomware while the execution time of screen-locker ransomware to lock the victim's desktop screen completely is 302.8 seconds on average, demonstrating that Peeler can also detect screen-locker ransomware at an early stage before locking a victim's system.
- We evaluate Peeler (without new training) against new and unseen ransomware (34 samples) from more than eight families, including crypto, screen-locker, and general malware samples. Peeler successfully detected all eight samples from crypto ransomware families and nine out of ten samples from screen-locker ransomware families. Additionally, Peeler also detected 9 out of 16 samples from general malware.

[1]Windows OS is becoming the most attractive targets for ransomware writers, i.e., 87% of the existing ransomware target Windows [23]

II. Key characteristics to detect ransomware

Peeler exploits the differences in system behavioral characteristics between ransomware and benign applications. We collected kernel-level events generated by diverse ransomware samples and benign applications and analyzed those events to gain insights into the unique and inherent system behaviors of ransomware. This section explores those characteristics in detail.

A. File I/O patterns

Our analysis on collected events reveals that crypto ransomware samples typically encrypt a user's file by performing the following four steps: access the file (*access*), read the content of the file (*read*), write the encrypted content to a temporary memory or new file (*write*), and overwrite/delete the user's original file (*overwrite/delete*). For example, Figure 1 shows a sequence of file I/O events from a variant of Cerber. These events align with the observed four file I/O steps as follows: 1) in the *access* step, the ransomware sample accesses a file (*D_186.wav*) with the `FileCreate` event; 2) in the *read* step, the ransomware sample reads the content of *D_186.wav* with the two `Read` events; 3) in the *write* step, the ransomware sample writes the encrypted content to the same file with the two `Write` events; and 4) in the *overwrite* step, the file is finally renamed with the `Rename`, `FileDelete`, and `FileCreate` events. As shown in Figure 1, the `FileDelete` operation removes the content of the original file *D_186.wav* and `FileCreate` assigns a new name *2O8nlobpEl.8cbe*. We note that the `File Key` remains the same for all events even though the original file's name and extension are changed.

PID	TID	Prov.	Event	Timestamp	File Key	Event-Attributes
XXXX	XXXX	File	FileCreate	13:25:11.325730	FFFFB203B649C160	\Desktop\Audio\D_186.wav
3496	1512	File	Read	13:25:11.325784	FFFFB203B649C160	60 395520
3496	1512	File	Read	13:25:11.325818	FFFFB203B649C160	4096 3942436
3496	1512	File	Write	13:25:11.327298	FFFFB203B649C160	110 0
3496	1512	File	Write	13:25:11.327301	FFFFB203B649C160	256 0
3496	1512	File	Rename	13:25:11.327499	FFFFB203B649C160	1512
XXXX	XXXX	File	FileDelete	13:25:11.327652	FFFFB203B649C160	\Desktop\Audio\D_186.wav
XXXX	XXXX	File	FileCreate	13:25:11.327654	FFFFB203B649C160	\Desktop\Audio\2O8nlobpEl.8cbe

Fig. 1: File I/O events generated by Cerber ransomware.

We observe that most crypto ransomware samples follow similar steps to lock a victim's files. However, the locking strategy adopted by each ransomware sample can be different. For example, some families lock a file without creating a temporary file, whereas others choose to lock a file via a temporary file; some deletes the original file, whereas others decide

to overwrite. We perform numerous trials and experiments and observe four *generic* file I/O patterns that characterize behaviors of most crypto ransomware families. We briefly summarize our findings as follows.

1) **Memory-to-File with Post-Overwrite:** As shown in Figure 1, some crypto ransomware samples directly overwrite a user's file with its encrypted data without creating a new file. The I/O events pattern observed is to overwrite the encrypted data to the original file and then rename its file name. This pattern can be observed in the following ransomware families: Cerber, Keypass, Telsacrypt, and Gandcrab.
2) **Memory-to-File with Pre-Overwrite:** This file I/O events pattern is similar to "Memory-to-File with Post-Overwrite" except that the original file is first renamed, and then the encrypted data is overwritten to that file. This pattern can be observed in samples from Locky ransomware family. Figure 2 shows a sequence of file I/O events generated from a sample in the Locky family. We can see that the original file is first renamed.
3) **File-to-File with Delete:** Some crypto ransomware samples create a new file and copy the encrypted data of the original file to the new file instead of overwriting the original file itself. After completing the copy process, the original file is finally deleted. This pattern can be observed in the following families: InfinityCrypt, Dharma, Malevich, Sage, and Syrk. Figure 3 shows a sequence of file I/O events generated from a sample in the InfinityCrypt family. We can see that the ransomware sample reads the file with the file key (`FFFFB203AFD146F0`) and writes it (in an encrypted form) to the file with the file key (`FFFFB203AFD14160`). After copying all the file content to the new file, the original file with the file key (`FFFFB203AFD146F0`) is deleted.
4) **File-to-File with Rename and Delete:** This file I/O events pattern is similar to "File-to-File with Delete" except that the new file is renamed after copying the encrypted data of the original file to the new file. This pattern can be observed in samples from the WannaCry ransomware family. Figure 4 shows a sequence of file I/O events generated from a sample in the WannaCry family. After copying all the file content to the new file, the file is renamed from *nasa.txt.WNCRYT* to *nasa.txt.WNCRY*.

PID	TID	Prov.	Event	Timestamp	FileKey	Event attributes
8040	6384	File	Create	13:41:53.709433	FFFFA9028BF1A3D0	\Desktop\Audio\ D_1025.wav 0 7
8040	6384	File	Rename	13:41:53.709535	FFFFD08F7AAE0700	FFFFA9028BF1A3D0 6384
XXXX	XXXX	File	FileDelete	13:41:53.709844	FFFFD08F7AAE0700	\Desktop\Audio\D_1025.wav
XXXX	XXXX	File	FileCreate	13:41:53.709846	FFFFD08F7AAE0700	\Desktop\Audio\BDWCJ979-P6FQ.thor
8040	6384	File	Read	13:41:53.710877	FFFFD08F7AAE0700	FFFFA9028BF1A240 91484 395520
8040	6384	File	Write	13:41:53.710979	FFFFD08F7AAE0700	FFFFA9028BF1A240 91484 0
8040	6384	File	Write	13:41:53.710983	FFFFD08F7AAE0700	FFFFA9028BF1A240 91484 395776
8040	6384	File	Write	13:41:53.711170	FFFFD08F7AAE0700	FFFFA9028BB2BB80 94208 393283

Fig. 2: File I/O events generated by Locky ransomware.

We show that most families from crypto ransomware follow one of the four file I/O patterns above described.

B. Application process tree

Screen-locker ransomware does not lock user files but instead lock users' screens. Hence, we failed to observe the file I/O patterns described in Section II-A from screen-locker ransomware. However, we observed screen-locker ransomware's behaviors that can be distinguishable from benign applications.

PID	TID	Prov.	Event	Timestamp	File Key	Event-Attributes
XXXX	XXXX	File	FileCreate	16:42:54.884727	FFFFB203AFD146F0	\Desktop\Audio\D_186.wav
XXXX	XXXX	File	FileCreate	16:42:54.926901	FFFFB203AFD14160	\Desktop\Audio\D_186.wav.0A57BC83D
3896	2720	File	Read	16:42:54.927172	FFFFB203AFD146F0	FFFF87040DE3E5F0 4096 395520
4	96	File	Read	16:42:54.927626	FFFFB203AFD146F0	FFFF87040DE3E5F0 126976 394243
3896	2720	File	Write	16:42:54.927801	FFFFB203AFD14160	FFFF87040DE3F400 4096 395776
4	96	File	Read	16:42:54.928425	FFFFB203AFD146F0	FFFF87040DE3E5F0 8192 394243 0
3896	2720	File	Write	16:42:54.928607	FFFFB203AFD14160	FFFF87040DE3F400 4096 395776
3896	2720	File	Delete	16:42:54.932136	FFFFB203AFD146F0	FFFF87040DE17360

Fig. 3: File I/O events generated by InfinityCrypt ransomware.

PID	TID	Prov.	Event	Timestamp	FileKey	Event attributes
XXXX	XXXX	File	FileCreate	10:40:44.580511	FFFFB203B69B36F0	\Desktop\FPM code\nasa.txt
3000	2556	File	Read	10:40:44.580523	FFFFB203B69B36F0	FFFF87040B60C460 8 395520
3000	2556	File	Read	10:40:44.580560	FFFFB203B69B36F0	FFFF87040B60C460 4096 394243
4	3008	File	Read	10:40:44.580705	FFFFB203B69B36F0	FFFF87040B60C460 126976 394243
XXXX	XXXX	File	FileCreate	10:40:44.580953	FFFFB203B69B3160	\Desktop\FPM code\nasa.txt.WNCRYT
3000	2556	File	Write	10:40:44.581134	FFFFB203B69B3160	FFFF87040B60E850 8 395776
3000	2556	File	Write	10:40:44.581185	FFFFB203B69B3160	FFFF87040B60E850 4 395776
3000	2556	File	Write	10:40:44.581197	FFFFB203B69B3160	FFFF87040B60E850 256 395776
3000	2556	File	Rename	10:40:45.805853	FFFFB203B69B3160	FFFF87040B60ACF0 2556
XXXX	XXXX	File	FileDelete	10:40:45.805994	FFFFB203B69B3160	\Desktop\FPM code\nasa.txt.WNCRYT
XXXX	XXXX	File	FileCreate	10:40:45.805996	FFFFB203B69B3160	\Desktop\FPM code\nasa.txt.WNCRY
3000	2556	File	Delete	10:42:11.557112	FFFFB203B69B36F0	

Fig. 4: File I/O events generated by Wannacry ransomware.

Applications can spawn one or more processes if it is needed. 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 screen-locker ransomware typically spawns many child processes compare to that of benign applications because screen-locker ransomware should create many processes to perform malicious tasks (e.g., connecting to C&C servers, modifying the Windows registry, hiding files/extensions) in parallel, resulting in large size of the application process tree. Figure 5 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.

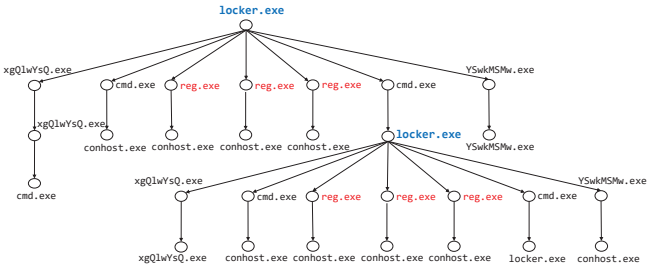

Fig. 5: Application process tree of VirLock.

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 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 Figure 5, three child processes (*reg.exe* in red color) perform the actual registry modifications. The corresponding command line execution is shown at the bottom of Figure 5.

In contrast, Figure 6 shows the application process trees of six benign applications such as `Chrome`, `Adobe Acrobat Reader`, `MS Visual Studio 2019`, `MS Office 365 ProPlus`, `Spotify`, and `MS Outlook`. Most of these applications create far less number of spawned processes compare to the screen-locker ransomware.

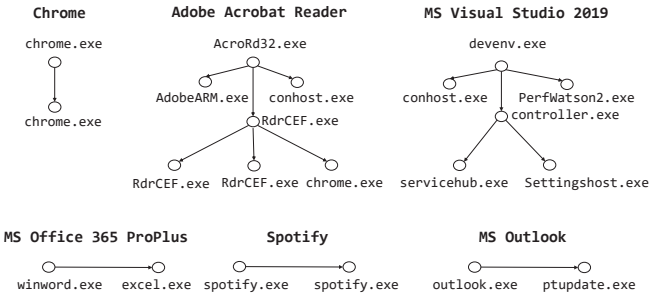

Fig. 6: Application process trees of benign applications.

Our observations on the behaviors of 104 screen-locker ransomware samples from 9 ransomware families show that more than 90% of samples generate a large application process tree. A screen-locker 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, significantly smaller to that generated by a screen-locker ransomware sample.

C. Causality relationships between system events

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

TABLE I: 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 I, 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 7 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.

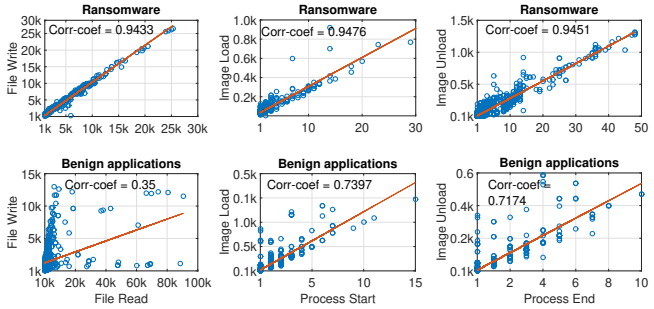

Fig. 7: Correlations among some event types for ransomware and benign applications.

III. System Design

Peeler is designed to minimize the overall damage by ransomware attacks using two different detection modules. Our design principle is to detect crypto ransomware attacks as early as possible because the number of encrypted files (i.e., the victim's damage) can be increased if the crypto ransomware attack detection is delayed. Therefore, for crypto ransomware, our design principle is to detect it immediately by using a simple pattern matching first as soon as the first file is encrypted. We believe that the sacrifice of one file is our best effort scenario in the view of dynamic analysis because Peeler leverages file I/O event patterns that can be generated when a file is encrypted by crypto ransomware. In contrast, we propose a different approach based on machine learning models to detect ransomware attacks concerning screen-locker ransomware (or sophisticated crypto ransomware that is not detected by a pattern matching module) because we need to analyze such cases with additional features carefully.

A. Overview

Figure 8 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.

Peeler has two main ransomware detection modules: 1) file I/O pattern matcher and 2) machine learning-based classifier.

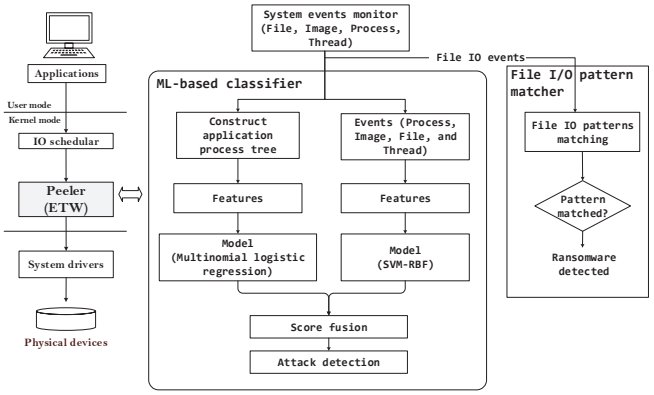

Fig. 8: Overview of Peeler.

The file I/O pattern matcher takes *kernel-level* file I/O events as input and looks for suspicious file I/O patterns that can be shown for ransomware. Once a pattern that characterizes ransomware behavior is detected, Peeler generates an alert. Machine learning-based classifier module extracts features from application process tree and system events providers (`Process`, `Image`, `File`, and `Thread`). The application process tree features are used to build a multinomial logistic regression model, whereas the system event features are used to build a Support Vector Machine (SVM) model. For detection, the scores from both classification models are fused as an ensemble approach, and then detection is performed.

Algorithm 1 enlists all the key steps required for ransomware detection. A window size W and a list of regular expressions (RE_{crypt}) are given as inputs to Algorithm 1. As shown in Figure 8, the system events monitor collects kernel events and forwards them to two modules (the file I/O pattern matcher and the machine learning-based classifier) in parallel. There are two differences between these modules: 1) file I/O pattern matcher consumes events generated from the `File` provider to identify suspicious file I/O patterns for focusing on crypto ransomware alone. Whereas ML-based classifier utilizes events from all four providers (`File`, `Process`, `Image`, and `Thread`) and application process tree features to detect both crypto and screen-locker ransomware. 2) File I/O pattern matcher performs analysis on a per-event basis (Step 2), whereas ML-based classifier accumulates events for W seconds (Step 7) and then processes them in a batch (Step 8). If both modules do not detect suspicious activities by ransomware, Peeler continuously monitors the system (Step 11).

B. System events monitor

Peeler provides a module called *system events monitor*, which is used to collect kernel events in real-time. Peeler relies on the following event providers [24]: `Process`, `Image`, `File`, and `Thread`. The data obtained by the system events monitor are in the form of a continuous sequence of events t_i. Each event has a schema describing the type of data contained in the event payload. An event is represented as:

Algorithm 1 Overall Process of Peeler

Input: W and RE_{crypt}
1: **while** true **do**
2: System events monitor receives an $event$ from ETW.
/* **Input events to both detectors in parallel.***/
3: **if** event is from `File` provider **then**
4: *CryptoMatcher* = FileIOPatternMatcher(*event*, RE_{crypt})
5: **if** *CryptoMatcher* **then**
6: Raise the alert and halt the process using *PID*.
7: *IncomingEvents* = Accumulate all events in a W seconds window.
8: *MLClassifierLabel* = ML-basedClassifier(*IncomingEvents*)
9: **if** *MLClassifierLabel* **then**
10: Raise the alert and halt the process using *PID*.
11: **else** Keep on monitoring the system.

$$t_i =< PID, TID, Prov., EType, E_{attrs} >,$$

where

PID is a process identifier,

TID is a thread identifier corresponding to the process PID,

$Prov.$ is a provider name,

$EType$ is an event name,

E_{attrs} is a set of attributes of the event E_{name}.

Our system events monitor leverages a tool called Event Tracing for Windows[2] (ETW) [24] for collecting events. ETW is an efficient kernel-level tracing solution that allows us to capture and consume system events in real-time. ETW consists of three main components : controller, consumer and provider. The controller starts and stops event tracing sessions. Each provider has its own set of events. Consumers retrieve events from providers. The providers and events used in Peeler are listed in Table II.

TABLE II: 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
File	Read, Write	PID, TID, Prov., Event, Timestamp	FileKey, FileObject, IoSize
	Rename, Delete	PID, TID, Prov., Event, Timestamp	FileKey, FileObject
	FileCreate, FileDelete	PID, TID, Prov., Event, Timestamp	FileObject, FileName
Thread	Start, End	PID, TID, Prov., Event, Timestamp	ParentId
Image	Load, Unload	PID, TID, Prov., Event, Timestamp	ImageSize, FileName

To implement the system events monitor in Peeler, we used an open-source project *krabsetw* [25], which is a C++ library that simplifies interactions with ETW. We modified *krabsetw* to collect the events only needed for Peeler. The events collected are used for file I/O pattern matcher and machine learning-based classifier in parallel.

C. File I/O pattern matcher

To effectively detect suspicious file I/O patterns, Peeler relies on regular expressions – a sequence of suspicious file I/O events is represented by a regular expression RE. Table III shows file I/O events and abbreviations used in regular expressions. File I/O patterns and corresponding regular expressions

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

are presented in Table IV. In the regular expression, the $*$ symbol represents zero or more repetitions of the immediately preceding sub-expression; the $+$ symbol represents one or more repetitions, while the ? symbol represents the zero or one repetition of the preceding. For example, the expression of WR^* indicates that a `Write` operation might be followed by zero or more `Read` operation(s).

TABLE III: File I/O events.

File I/O Event	Abbr. in regular expression
Read, Write	R, W
Rename, Delete	N, D
FileCreate, FileDelete	C, D
Create	C

TABLE IV: Regular expressions for the file I/O patterns of crypto ransomware.

File I/O patterns	Regular expression
Memory-to-File with Post-Overwrite	$C[R^+W^+R^*]^+NDC$
Memory-to-File with Pre-Overwrite	$CNDC[R^+W^+R^*]^+$
File-to-File with Delete	$C^+[R^+C^?W^+R^*]^+D$
File-to-File with Rename and Delete	$C^+[R^+C^?W^+R^*]^+NDC$

Algorithm 2 FileIOPatternMatcher.

Input: *event* = <*PID*, *EType*, *FileObject*, *FileName*, *FileKey*> and RE_{crypt}
Output: *Benign* and *Ransomware*

Stage 1: Preparing the data
if EType is 'FileCreate' **then**
  if FileObject and FileName are newly observed **then**
    Create events list *FileEventsList* for the newly observed file.
    Add *event* to *FileEventsList*.
else add *event* to respective file's *FileEventsList*.

Stage 2: File I/O patterns detection
if the number of unique ETypes in the *FileEventsList* is equal or greater than four **then**
  Extract Etypes sequence from *FileEventsList*.
  if Etypes sequence matched to at-least one of the regular expressions in RE_{crypt} **then**
    Flag the process PID in *event*.
    Ransomware = True
else continue

Stage 3: Filtering false positives.
if *FilePath* are different in *FileEventsList*'s events **then**
  Benign = True
if PIDs in *FileEventsList*'s events are not same **then**
  Benign = True
if file extension in *FileEventsList*'s events is in ['.bak', '.log', '.TMP', '.jtx', '.journal', '.tmp', '.db-journal', '.dbx-journal'] **then**
  Benign = True

Algorithm 2 is composed of three main stages for detecting crypto ransomware. The input to the Algorithm is events from file I/O provider. An event is specified as *event* = <*PID*, *EType*, *FileObject*, *FileName*, *FileKey*> and with a set of regular expressions, RE_{crypt}. *PID* and *EType* refer to the process identifier and event type, respectively. *FileObject* is a unique key assigned to every file, whereas *FileName* contains the name of a file with full path. *FileKey* is an identifier that is used to find the file on which an event is performed. For example, the event schema for `Read` and `Write` (see Table II) does not contain *FileName* attribute. In order to obtain the file name, the *FileKey* attribute of the `Read`/`Write` event is matched with the *FileObject* attribute of the `FileCreate` event.

1) Preparing the data: The continuous stream of incoming file I/O events are first processed such that they are added to a separate list based on the file they are operating on. Whenever a user's file is accessed (i.e., `FileCreate` event is received), an events list *FileEventsList* is created and the corresponding event (`FileCreate`) with respective attributes is added to that list. All subsequent file I/O operation events (such as `Read`, `Write`, `Rename`, `Delete`, `FileCreate`, and `FileDelete`) on that file are added to its *FileEventsList* (Step 1 – 6 in Algorithm 3).

However, it is challenging to accurately add all file I/O events corresponding to a user's file to its respective *FileEventsList*. Because a single file may have multiple *FileObject* keys and vice versa. For example, as shown in Figure 3, the *FileObject* keys for `Read` and `Write` events on the file `D_186.wav` are not the same, but the actual file on which the operations are performed is the same. To ensure that all events associated with a file are fully recorded, Peeler leverages both *FileObject* and *FileName* attributes to accurately identify a user's file.

2) File I/O pattern matcher: Suspicious file I/O patterns are detected by analyzing the sequence of events in the list *FileEventsList* corresponding to a file. If the number of unique *EType*s is equal to or more than four (Step 6 in Algorithm 3), the events in the list are analyzed for suspicious patterns. Peeler enforces the *four unique events* constraint to reduce unnecessary computational overhead because at least four unique event types are required to encrypt a user file (see Figures 1–4). As presented in Table IV, suspicious file I/O patterns for crypto ransomware are represented in the form of regular expressions. If the sequence of incoming events in *FileEventsList* is matched with one of those regular expressions, Peeler reports this sequence as the evidence of crypto ransomware activity for a security warning.

We note that the **Memory-to-File** operations can be distinguished from the **File-to-File** operations by using the *FileObject* keys. When the same file is overwritten, we observe that the *FileObject* key remains the same for all file I/O events, e.g., the sequence of events to encrypt the file *D_186.wav* are shown in Figure 1 for Cerber ransomware. For **File-to-File** operations, multiple files associated with *FileObject* keys are needed. That is, in this case, the file I/O events from multiple files are accumulated into the same *FileEventsList* list.

3) Filtering false positives: We observe that some benign applications may generate ransomware-like file I/O patterns. For example, some benign applications may overwrite Windows OS's Activation Tokens file (*tokens.dat*), which can lead to false positives because it generates file I/O events that look similar to **Memory-to-File** patterns. We applied the following three heuristics to reduce the possibility of such false-positive cases (see Stage 3 in Algorithm 2):

- If *FilePath*s in *FileEventsList* are not directing to the same file, this is ignored because the file encrypted must be in

the same location.

- All the file I/O events must be performed by the same process (i.e., the same *PID*) with exception for the *system* and *explorer.exe* processes. For example, event lists in Figures 3 and 4 show that *system* process (*PID* = 4) is involved. Similarly, *explorer.exe* is also system process that may assist other processes in performing tasks.
- Peeler ignored the file I/O operations on temporary files. In our current implementation, such temporary files can be simply identified with their file extensions, such as *bak*, *tmp*, *log*, *TMP*, *jtx*, *journal*, *db-journal*, and *dbx-journal*.

D. Machine learning-based classifier

Peeler uses two different machine learning-based (ML-based) classifiers with application process tree features, and system event features to detect ransomware samples that cannot be detected by the file I/O pattern matcher. Algorithm 3 enlists all the steps for ML-based classifiers. 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 (Step 1–4 in Algorithm 3). The built machine learning models are then used to detect ransomware attacks (Step 5–7 in Algorithm 3).

Algorithm 3 ML-basedClassifier.

Input: *IncomingEvents*
Output: *Benign* and *Ransomware*

Stage 1: Feature sets extraction and model building.
1: FV_{MLR} = extract application process tree features from *IncomingEvents* (see Table V).
2: FV_{SVM} = extract system events features from *IncomingEvents* (see Table V).
3: *M1* = Build ML Multinomial logistic regression model with FV_{MLR} feature set.
4: *M2* = Build ML SVM-RBF model with FV_{SVM} feature set.

Stage 2: Attack detection.
5: Extract feature set vectors for test samples.
6: Test FV_{MLR} and FV_{SVM} features on models *M1* and *M2*, respectively.
7: Fuse scores from models and provide the class label (either `benign` or `ransomware`) as output.

1) Building the first classifier with the application process tree features: Windows applications are typically descended from the parent process called *explorer.exe*. Therefore, all applications' processes share *explorer.exe* as their parent process. As discussed in Section II-B, we observed that several screen-locker samples spawn a significantly larger number of child processes compared to benign applications. Based on this observation, Peeler constructs a machine learning model using the application process tree features. Peeler specifically extracts the following features from the application process tree: the number of processes, the number of unique processes, the number of threads created by processes, the maximum depth level of the tree, and the number of leaf nodes in the tree. For the first classifier using the feature set FV_{MLR} (see Table V), we selected a multinomial logistic regression (MLR) model because we do not assume linear relationships among the features in FV_{MLR} [26] and MSR produces the best performance with those features.

2) Building the second classifier with the system event features: As discussed in Section II-C, 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 (see Table V) and train 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}.

TABLE V: Feature extraction.

Feature set	Feature	Model
FV_{MLR}	# of processes Sum of threads from processes Maximum depth level of process tree # of leaf nodes # of unique process names	MLR
FV_{SVM}	# 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	SVM-RBF

3) Attack detection: Peeler uses two classification models (MLR and SVM-RBF), and finally decides the classification outcome by fusing their scores. We note that MLR and SVM-RBF are constructed with different feature sets – MLR is trained with FV_{MLR} while SVM-RBF is trained FV_{SVM} (see Table V). The scores from the two models are fused by taking their average for detection.

IV. Dataset collection

We aimed to collect a ransomware dataset containing diverse ransomware families rather than similar ransomware variants. Therefore, we collected 206 ransomware samples from 43 families (see Section IV-A). Also, we collected benign applications (see Section IV-B) containing crypto-like and screen-locker-like benign applications (see Appendix A) to evaluate Peeler's robustness against false positives.

A. Ransomware

We first obtained 3,040 malware samples from VirusTotal [27], malware repository [28], malwares [29], 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 [16]. 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 [30]. Therefore, we finally used 206 active ransomware samples that perform their activities correctly.

1) User environment and data collection: We used VirtualBox 6.1 [31] 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 60 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. In total, 206 ransomware samples from 43 distinct ransomware families (see Table VI) were collected. Out of 43 families, 34 (102 samples) were from crypto, and 9 (104 samples) were from screen-locker types of ransomware.

TABLE VI: Ransomware families, types, and samples.

no.	Family	Type	Samples	no.	Family	Type	Samples
1	Cerber	Crypto	33	23	Petya	Crypto	1
2	Sodinokibi	Crypto	14	24	Satana	Crypto	1
3	GoldenEye	Crypto	12	25	Shade	Crypto	1
4	Sage	Crypto	5	26	Syrk	Crypto	1
5	Locky	Crypto	5	27	TeslaCrypt	Crypto	1
6	Dharma	Crypto	3	28	ucyLocker	Crypto	1
7	dotExe	Crypto	3	29	Unlock92	Crypto	1
8	Troldesh	Crypto	1	30	Vipasana	Crypto	1
9	WannaCry	Crypto	3	31	Xorist	Crypto	2
10	Da Vinci Code	Crypto	1	32	Malevich	Crypto	1
11	CryptoShield	Crypto	1	33	Jigsaw	Crypto	1
12	CryptoWire	Crypto	1	34	Adobe	Crypto	1
13	District	Crypto	1	35	Virlock.Gen.5	Screen	83
14	Gandcrab	Crypto	1	36	LockScreen.AGU	Screen	12
15	GlobeImposter	Crypto	1	37	Alphabet	Screen	2
16	Hexadecimal	Crypto	1	38	EgyptianGhosts	Screen	1
17	InfinityCrypt	Crypto	1	39	Lockey-Pay	Screen	1
18	IS (Ordinpt)	Crypto	1	40	Blue-Howl	Screen	1
19	Keypass	Crypto	1	41	ShellLocker	Screen	1
20	Lockcrypt	Crypto	1	42	DerialLock	Screen	1
21	Pack14	Crypto	1	43	Trojan.Ransom	Screen	1
22	PocrimCrypt	Crypto	1		-		
Total samples: 206, crypto = 102, screen-locker = 104							

2) Diversity in our dataset: Table VI 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 [32], [16], 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 [32]. Scaife et al. [16] 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. It is because the core behavioral traits shown by crypto ransomware in encrypting data attack does not change from one variant to the other or from one family to the other. Since our study covered more than eight times the number of families from previous study [33], and more than two times the number of families covered in studies [15], [16] and there was not much diversity within families, there was little need to collect additional samples.

B. Benign applications

We also collected the dataset for popularly used applications that are typically installed on a benign user's computer. 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: 1) benign applications with file I/O patterns resembling crypto ransomware, 2) benign applications with process tree resembling screen-locker ransomware, and 3) commonly used benign applications.

1) Benign applications resembling crypto ransomware: Benign applications performing encryption or compression might generate file I/O patterns similar to crypto ransomware that could result in false positives. To evaluate Peeler against, we collected data from several crypto-like benign applications, listed in Appendix A. There are key differences in file I/O patterns generated by encryption/compression tools compared to crypto-ransomware. Firstly, unlike ransomware incurring a massive number of repeated file I/O patterns, the encryption/compression tools operate on a limited number of files only. Secondly, the original user files remain intact, i.e., not overwritten or deleted, even after compression or encryption is performed. It is, therefore, doubtful that benign applications show ransomware file I/O patterns. Thirdly, unlike crypto ransomware that encrypts user files arbitrarily, benign applications create a dedicated process that needs sophisticated inputs from the OS to complete the task. For instance, the compression tool *7-zip* takes several parameters to specify target files. Each tool in Appendix A is run twice – for compression and decompression, on a given set of files to collect data.

2) Benign applications resembling screen-locker ransomware: We collected the dataset containing applications that spawn many child processes to evaluate Peeler's robustness against false positives. We found that certain benign applications such as Pycharm and Visual Studio create many spawned processes that may resemble screen-locker ransomware. The list of benign applications is shown in Appendix B. We collected data by running each application individually.

3) Commonly used benign application: We also collected user's system usage data under normal conditions while interacting with commonly used applications. A user runs many different applications at the same time. For example, the user read 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. Our goal here is to analyze system events generated in an interleaved manner from commonly used benign applications. The collected data is for around 12 hours of computer usage. During data collection,

the user interacted with multiple applications from the list of benign applications in Appendix A.

V. Evaluation

We demonstrate Peeler's performance in detection accuracy, detection time, and CPU/memory overheads.

A. Experimental setup

For evaluation, we used the dataset described in Section IV. For training, 20% of both screen-locker ransomware and benign applications randomly are selected. We used a small training dataset (only 20% of the entire dataset) for the following reasons: 1) the size of a training set is typically limited in the real world; 2) we want to increase the number of testing samples as many as possible for making Peeler robust against unseen ransomware samples; 3) we want to reduce the size of model for quick training. All the remaining ransomware and benign samples (i.e., 80% of the total samples) are used for testing purposes.

B. Detection accuracy

Table VII shows the summary of Peeler's detection accuracy. Overall, Peeler achieved 99.52% accuracy with a false positive rate of only 0.58%. Since we used only 20% of applications selected randomly for training, all performance statistics are averaged after 100 runs to mitigate biases in training and testing datasets. Similarly, Peeler achieved high precision and F1 score, which are greater than 99%.

TABLE VII: Peeler's detection accuracy.

Metric	Accuracy(%)
Accuracy	99.52%
True positive rate	99.63%
True negative rate	99.42%
False negative rate	0.37%
False positive rate	0.58%
Precision	99.41%
Recall	99.63%
F1 score	99.52%

1) 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 VIII):

TABLE VIII: Peeler's false positive analysis for different types of benign applications.

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 quite similar to those of typical crypto ransomware, but they do not restrict access to files via encryption, unlike crypto ransomware.

Table VIII shows that Peeler correctly detects 98.27% with a false positive rate of 1.72% even against crypto ransomware-like benign applications.

Screen-Locker-like benign applications. As described in Section II-B, screen-locker ransomware typically spawns 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 Store (https://www.microsoft.com/en-us/store/apps/windows) (see Appendix B) and selected 18 applications showing such behaviors. Table IX presents the applications' process tree-related feature (FV_{MLR}) values of three representative benign applications showing such behaviors. We observe that Pycharm and Visual Studio spawned 140 and 46 child processes, respectively. Interestingly, Chrome spawns 42 processes, but the depth of its applications' tree is one, and the number of threads created by these processes is 1,480. We examined the results of Peeler with those 18 applications.

TABLE IX: Benign applications' process tree features.

Application	# processes	Depth	# leaf nodes	# unique processes	# threads
Pycharm	140	4	70	11	993
Visual Studio	46	4	29	21	568
Chrome	42	1	41	2	1,480

Table VIII shows that Peeler correctly detected 99.5% with a false positive rate of 0.31% even though these benign applications show screen-locker-like behavior in terms of spawned processes. The reason for this detection is because Peeler also considers the other set of features (FV_{SVM}), which are related to system events.

Commonly 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 Sectoin IV-B.

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%.

2) False negative analysis: The false negative rate for ransomware detection is another important metric to evaluate the effectiveness of Peeler. Table X shows that the overall false negative rate is above 0.37%. The false negative rates for crypto and screen-locker ransomware are 0% and 0.5%, respectively.

3) Model-specific detection accuracy.: We constructed Peeler by building multiple detection components (file I/O pattern matcher and machine learning-based classifier). Here

TABLE X: Peeler's false negative rate analysis.

Ransomware	TPR (%)	FPR (%)	FNR (%)
Crypto	100	0.8	0
Screen-locker	99.50	0.31	0.50
All ransomware	99.63	0.58	0.37

we show the effectiveness of each component in detecting ransomware. Table XI and XII show the evaluation results of both components, respectively.

TABLE XI: File I/O pattern matcher's performance.

Ransomware	TPR (%)	FPR (%)	FNR (%)	Prec. (%)	Rec. (%)	F1 score (%)
Crypto	95.19	0	4.81	100	95.19	97.53
Screen-locker	33.33	0	66.66	100	33.33	50.00
All ransomware	65.50	0	34.5	100	65.50	79.15

Table XI shows that the file I/O pattern matcher achieves the detection rate of 95.19% with the false rate of 0% in detecting crypto ransomware; however, it is not effective in detecting screen-locker ransomware.

Table XII shows that the machine learning-based classifier works well in detecting both crypto and screen-locker ransomware. The machine learning-based classifier achieves more than 98% detection accuracy with less than 2% false rate in detecting both types of ransomware attacks. The high detection rate of the machine learning-based classifier can be attributed to the feature vectors (FV_{MLR} and FV_{SVM}). However, to boost the detection time, we can first apply the file I/O pattern matcher and then use the machine learning-based classifier only when the file I/O pattern matcher fails to detect.

TABLE XII: Machine learning-based classifier's performance.

Ransomware	TPR (%)	FPR (%)	FNR (%)	Prec. (%)	Rec. (%)	F1 score (%)
Crypto	98.45	1.84	1.54	98.24	98.45	98.29
Screen-locker	99.5	0.31	0.50	99.68	99.50	99.59
All ransomware	99.05	1.9	0.94	98.19	99.05	98.59

Each component can work as an alternative and complementary detection method to the other component. For instance, the file I/O pattern matcher failed to detect some crypto ransomware samples such as Hexadecimal, Cryptowire, CryptoShield, and CryptoLock, but the machine learning-based classifier successfully detected them.

C. Detection time

For crypto ransomware, the detection time refers to the time interval between the end of the detection and the end of encryption on a file by a process, that is, how long it takes Peeler to detect the ransomware attack after a ransom sample encrypts a file. We excluded the time to be taken for the file encryption because that time can be varied depending on the file size.

We measured the detection time of Peeler against 102 samples from 34 different crypto ransomware families. Figure 9 shows that Peeler can detect over 70% of samples within 115 milliseconds with the mean time of 115.3 milliseconds, demonstrating that Peeler outperforms existing crypto ransomware detection solutions in detection time. Peeler can promptly detect crypto ransomware with a simple, regular expression-based pattern matching, unlike other existing solutions relying on complicated file activity (e.g., identifying encryption operations based on entropy computation) analysis or machine learning models.

These detection time results demonstrate the superiority of Peeler compared with existing crypto ransomware detection solutions. Cryptolock [16] detects crypto ransomware after 10 files are encrypted. Similarly, Mehnaz et al. [12] presented a solution which takes on average 8.87 seconds to detect malicious processes.

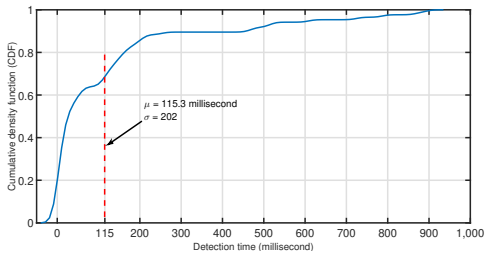

Fig. 9: CDF of the ransomware detection time for 102 crypto ransomware samples.

We also measured the detection time of Peeler against 104 samples from 9 different screen-locker ransomware families. For screen-locker ransomware, the detection time refers to the time interval between the end of the detection and the start activity by a ransomware process. Figure 10 shows that Peeler, on average, took 16.4 seconds to detect screen-locker ransomware while the execution time of screen-locker ransomware to lock user's desktop screen completely is 302.8 seconds on average, demonstrating that Peeler can detect screen-locker ransomware at a very early stage that prevents an attacker from locking a victim's system.

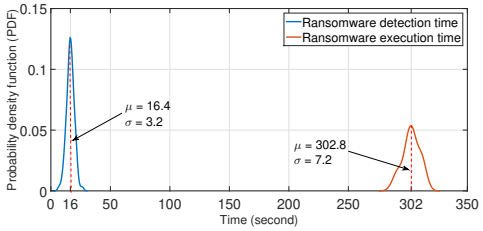

Fig. 10: Probability density function of detection times for 104 screen-locker ransomware samples with mean (μ) and standard deviation (σ).

D. CPU and memory overheads

We evaluate the performance of Peeler with respect to CPU and memory overheads. Since Peeler intercepts low-level kernel events and then analyzes them to detect ransomware attacks, its performance overheads in CPU and memory can typically be changed depending on the system's workload. For instance, we observe that if computationally intensive tasks are running, the overheads of Peeler inherently increase

because Peeler should frequently intercept a high number of kernel events generated from such computationally intensive processes.

TABLE XIII: CPU and memory overheads of Peeler.

Workload	CPU (%)		Memory (MB)	
	Mean	Std.	Mean	Std.
Low	2.7	2.0	9.8	0.3
Normal	4.9	2.2	9.8	0.4
High	15.8	3.9	11.3	1.9

Our experiments were conducted on a computing device equipped with two Intel Core(TM) i5-7300U (2.60GHz) CPUs and 8GB RAM, running 64-bit Windows 10 Enterprise edition operating system. Our CPU and memory usage results were measured based on this environment. We considered the three workload conditions as follows: Low, Normal, and High workload conditions. Under the low workload condition, Peeler is only running in the background and continuously collecting system events and writing them to a log file. Under the normal workload condition, we additionally performed the following tasks: 1) drafting an MS Word document; 2) using Chrome for browsing online material; and 3) reading a document using Adobe Acrobat Reader. Under the high workload condition, we additionally ran a CPU intensive algorithm as a background process. We note that an anti-malware program called CylanceProtect and default Windows services were continuously running in the background for all workload conditions.

Table XIII shows CPU and memory usage of Peeler. We observe that mean CPU usage and the standard deviation are 4.9% and 2.2%, respectively, under the normal workload condition.

Table XIII also shows that mean memory usage is around 9.8MB under the low or normal workload conditions, which is quite stable and has less variation (standard deviation is 0.4MB). Under the high workload condition, the average CPU usage of Peeler significantly increases to 15.8% with a standard deviation of 3.9% while its average memory usage slightly increases to 11.3MB memory with a standard deviation of 1.9MB.

E. Robustness against unseen ransomware families

To test Peeler against unseen ransomware samples, we additionally collected new and *unseen* ransomware samples after three months from the first experiments. A total of 34 samples from more than eight distinct unseen ransomware families are tested. We used the previously constructed Peeler without retraining. All samples tested in this experiment are manually verified from VirusTotal to confirm their family and type.

The detection results of Peeler against those samples are presented in Table XIV. Peeler was able to correctly detect all unseen crypto ransomware samples while the detection rate for screen-locker was degraded by about 9% compared to the detection rate in Table X. Interestingly, Peeler can also detect 9 out of 16 conventional malware samples even though those samples do not have ransomware capabilities. For example, our investigation revealed that *Backdoor.Generic* is a malware family that enables attackers to control infected computers remotely to create large groups of zombie computers (botnets), which are then used for malicious purposes without the user's knowledge. We surmise that those malware samples have some common behaviors that can be observed from ransomware.

TABLE XIV: Detection of *unseen* ransomware and malware.

Type	Family	File extension	Sample	Detection
Crypto	Matrix	`.CORE`	3	8/8
	Fox	`.FOX`	3	
	Crpren	`.af46`	1	
	MedusaLocker	`.encrypted`	1	
Screen-locker	VirLock.Gen.8	-	6	9/10
	Trojan.Killproc	-	2	
	VirLock.16	-	2	
Malware (non ransomware)	Backdoor.Generic	-	2	9/16
	Unknown	-	14	

VI. Discussion and Limitations

A. Comparison with existing ransomware detection solutions

As mentioned in Section I, there are many existing methods to detect ransomware attacks. However, since most existing solutions used their own dataset for evaluation, and their source code is not opened, we do not directly compare Peeler with those solutions. Alternatively, we compare Peeler with those solutions according to their experimental results reported in their papers. Table XV shows a summary of the comparison results.

TABLE XV: Comparison with existing approaches.

Method	TPR (%)	FPR (%)	Files lost	Screen-locker?	Samples/families	Real-time
Redemption [6]	100	0.8	5	×	677/29	×
CryptoLock [16]	100	0.03	10	×	492/14	×
UNVEIL [15]	96.3	0	-	✓	2121/ -	×
REDFISH [14]	100	-	10	×	54/19	×
RWGuard [12]	-	0.1	partial recovery	×	- /14	✓
Elderan [11]	93.3	1.6	-	×	582/11	×
CM&CB [34]	98	Vary	-	×	8/ -	×
RansHunt [35]	97	3	-	×	360/20	×
ShieldFS [9]	100	0.038	-	×	383/11	×
Peeler	**99.63**	**0.58**	**1**	✓	**206/43**	✓

We can see that the ransomware detection rates overall ranges from 93% to 100%. However, it is important to note that the number of ransomware samples/families used for evaluation is different for each approach. CryptoLock [16], Redemption [6], REDFISH [14], and ShieldFS [9] achieved 100% detection rates, but those solutions were tested on 14, 29, 19, and 11 ransomware families only, respectively. In contrast, Peeler was tested against 43 distinct ransomware families, including both crypto and screen locker ransomware, and still achieved a 99.63% detection rate. For the false positive rate with benign applications, Peeler achieved 0.58% FPR, which would be comparable with the other solutions.

For crypto ransomware detection, one of the most critical evaluation metrics is the number of user files lost before a ransomware sample is detected. Peeler detects ransomware immediately after a single file alone is encrypted, indicating that Peeler outperforms other solutions that reported the results in the number of lost files before detection.

From Table XV, we can see that only two solutions (Peeler and UNVEIL [15]) considered the detection of screen-locker ransomware. However, Peeler's detection time (16.4 seconds on average) would significantly be faster than UNVEIL. Although Kharaz et al. [15] did not report the exact detection time of UNVEIL, we surmise that UNVEIL would take longer detection time because it needs to capture screenshots of a victim's desktop periodically, and then compute the similarity between the captured images.

B. Secure implementation of Peeler

Peeler runs in a privileged kernel mode with administrative rights because it needs to collect kernel-level events from four main kernel providers (`File`, `Process`, `Image`, and `Thread`). Therefore, the integrity of Peeler can be securely protected against malicious processes running in the user mode. However, if we consider powerful attackers (e.g., rootkit-based ransomware) with the root privilege, we additionally need to consider deploying a kernel protection mechanism to protect Peeler against attackers. Recently, several techniques (e.g., real-time kernel protection (RKP) [36]) have been proposed to protect the kernel code and objects from unauthorized modifications. With such a secure kernel protection mechanism, we can implement Peeler as a Windows service running on the kernel.

C. Parameters optimization and extension to other computing environments

Peeler is flexible to adjust the window size parameter W for the machine learning-based classifier. However, the optimal selection of W should be carefully selected with two fundamental issues: detection accuracy and detection time. We empirically observed that there is a trade-off between detection accuracy and detection time. If we use a long time period for W, Peeler's detection accuracy would be improved, although the use of such a long time period results in more chances for ransomware to lock a victim's files or system. On the other hand, if we use a short time period for W, Peeler's detection accuracy would be degraded while Peeler's detection time can be reduced. Based on this trade-off relationship, we empirically set $W = 5$ seconds in all the experiments in this paper. However, W can be optimized according to conditions of the environment in which Peeler will be used.

In order to extend Peeler to other computing environments such as Linux or Android, two aspects are needed to be considered: platform-depended system events and key ransomware behavioral characteristics. The later remains platform-independent because the key behavioral characteristics of ransomware remain almost the same no matter which platform they target. However, system events for other computing environments need to be carefully analyzed to extend Peeler to these environments. For instance, the suspicious file I/O patterns and corresponding regular expressions should be updated to reflect platform-dependent system events.

D. Limitations

In our experiments, the tested benign applications are only a part of a massive number of benign applications. Therefore, we can still have a chance to encounter (unknown) benign applications that generate suspicious file I/O patterns that might result in false positives. For instance, when Windows OS creates a backup file `tokens.dat.bak` from the file of `tokens.dat`, we found that the sequence of kernel-level file I/O events generated from a benign process can lead to a false positive result from the file I/O pattern matcher. To avoid such false positive cases, Peeler currently uses a whitelist containing well-known system file extensions (such as `.bak`, `.log`, `.TMP`, `.jtx`, `.journal`, `.db-journal`, and `.dbx-journal`).

If the regular expressions used in Peeler are exposed to attackers, they can try to generate different file I/O patterns that are not matched to all the regular expressions. For example, a ransomware sample can perform unnecessary and dummy file operations such as `Rename` or `FileCreate` repeatedly to generate unknown file I/O patterns such as $CNDCNDC[R^{+}W^{+}R^{*}]^{+}$. Therefore, Peeler's regular expressions should be generally extended to cover such cases with dummy operations and should be confidentially protected. We note that the current regular expressions are the minimal representations for suspicious file I/O patterns. Moreover, a sophisticated attacker may misuse Peeler's three heuristics, such as the use of whitelist for system file extensions. Therefore, we need to consider developing more advanced and generic heuristics to prevent such adaptive attacks so that we deploy Peeler in real-world environments. Fortunately, we note that the machine learning-based classifier in Peeler can be effectively used to detect such adaptive ransomware attacks as a complement to the file I/O pattern matcher.

VII. Related work

We categorize the literature regarding ransomware detection into three groups: 1) crypto ransomware detection techniques that are mainly based on certain 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 to 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. Kharraz et al. [15] 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. [6] 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. [16] also presented a system called CryptoDrop that detect 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. [14] 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. Although the idea of using file I/O patterns is not new, Peeler is specifically designed to detect crypto ransomware's suspicious activities with kernel-level file I/O events to reduce the detection time while most of the existing solutions require high-level file I/O activities (e.g., encryption) translated from kernel-level file I/O events. Peeler only focuses on detecting suspicious kernel-level file I/O event sequences that should contain file operations (i.e., overwrite or delete) on the original file.

Another approach is to recover the files encrypted by crypto ransomware. Continella et al. [9] proposed ShieldFS that monitors file system operations to build an anomaly detection model using processes' system behaviors. When a process deviates from the model for benign processes, the file operations performed by the suspicious process are transparently rolled back. However, ShieldFS requires shadowing files whenever they are modified, thereby incurring a significant overhead in space. Kolodenker et al. [37] proposed a system called PayBreak, which intercepts the system's crypto functions, identifies the keys used for the functions, and stores them. The stored keys are used later to decrypt the files encrypted by ransomware attacks. However, PayBreack can decrypt files only for the ransomware families that use system provided crypto functions. Huang et al. [17] proposed FlashGuard, a ransomware-resistant Solid State Drive (SSD) system, which provides a recovery functionality at the firmware-level allowing quick and effective recovery from crypto ransomware without relying on explicit backups.

Most existing solutions have been developed, focusing on detecting crypto ransomware only. To the best of our knowledge, Peeler is the first unified system that works for both crypto and screen-locker ransomware. Peeler uses a hybrid approach that leverages both file I/O pattern-based rules and machine learning models for accurate and efficient detection of crypto and screen-locker ransomware. We note that our hybrid approach is different from UNVEIL [15] in that UNVEIL employs two completely different systems for detecting crypto and screen-locker ransomware, respectively, while Peeler's machine learning-based classifier is used to detect both ransomware types at the same time.

Machine learning based ransomware detection. RWGuard [12] is a machine learning-based crypto ransomware detection system. RWGuard 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. Unlike RWGuard, however, we additionally considered the causality relationships among system events and the applications' process tree properties to detect both crypto and screen-locker ransomware. Sgandurra et al. [11] 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. [38] and Al-rimy [21] proposed behavior-based machine learning models for ransomware detection. Hirano et al. selected five-dimensional features that were extracted both from ransomware and benign applications' I/O log files. Nieuwenhuizen [32] proposed another behavioral-based machine learning model using a feature set that quantifies the behavioral traits for ransomware's malicious activities. Cohen et al. [8] analyzed memory dumps taken from virtual machines where ransomware samples are executed. They developed machine learning algorithms based on the meta-features extracted from the memory analysis. The proposed machine learning model achieved an F-score of 97.6% with five ransomware families.

Decoy files based ransomware detection. Decoy techniques [12], [13], [9] have also been frequently proposed to detect ransomware attacks. For example, Gomez et al. [13] 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 real user files, sophisticated ransomware samples ignore decoy files [12]. Moreover, it is also unclear how those solutions would detect some ransomware families (e.g., Petya) that affect predefined system files only.

VIII. Conclusion and future work

We propose a new effective and efficient solution called Peeler to detect ransomware attacks using their system behaviors. Peeler is built on the I/O pattern matcher and machine learning models to improve the detection accuracy and reduce the detection time. Most crypto ransomware can be detected by the I/O pattern matcher efficiently; the crypto ransomware that cannot be detected by the I/O pattern matcher or screen-locker ransomware can be detected by the machine learning models more accurately.

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 99.52% accuracy with a false positive rate of only 0.58%. Moreover, Peeler is efficient in detecting crypto ransomware – over 70% of crypto ransomware samples can be detected within 115 milliseconds. Although Peeler's detection time (16.4 seconds on average) is relatively slower for screen-locker ransomware, it is still sufficient to detect it because it typically takes a longer time (302.8 seconds on average) to lock a victim's system entirely.

As future work, we plan to extend our approach to the detection of other types of malware (e.g., cryptojacking [39]). To achieve this, we also plan to leverage other providers from Windows such as `PowerShell` and `Network`, and analyze kernel-level events generated by new malware types.

REFERENCES

[1] *First Three Quarters of 2019: 7.2 Billion Malware Attacks, 151.9 Million Ransomware Attacks*. https://www.securitymagazine.com/articles/91133-first-three-quarters-of-2019-72-billion-malware-attacks-1519-million-ransomware-attacks.

[2] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions," *Computers & Security*, vol. 74, pp. 144–166, 2018.

[3] N. Popper, *Ransomware Attacks Grow, Crippling Cities and Businesses*. https://www.nytimes.com/2020/02/09/technology/ransomware-attacks.html.

[4] E. M. Lab, *The State of Ransomware in the US*. https://blog.emsisoft.com/en/34822/the-state-of-ransomware-in-the-us-report-and-statistics-2019/.

[5] S. Sivakorn, K. Jee, Y. Sun, L. Kort-Parn, Z. Li, C. Lumezanu, Z. Wu, L.-A. Tang, and D. Li, "Countering malicious processes with process-dns association.," in *Network and Distributed Systems Security*, 2019.

[6] A. Kharraz and E. Kirda, "Redemption: Real-time protection against ransomware at end-hosts," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 98–119, 2017.

[7] S. Poudyal, K. P. Subedi, and D. Dasgupta, "A framework for analyzing ransomware using machine learning," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1692–1699, 2018.

[8] A. Cohen and N. Nissim, "Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory," *Expert Systems with Applications*, vol. 102, pp. 158–178, 2018.

[9] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "ShieldFS: a self-healing, ransomware-aware filesystem," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 336–347, 2016.

[10] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, and R. Khayami, "Know abnormal, find evil: frequent pattern mining for ransomware threat hunting and intelligence," *IEEE transactions on emerging topics in computing*, 2017.

[11] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," *arXiv preprint arXiv:1609.03020*, 2016.

[12] S. Mehnaz, A. Mudgerikar, and E. Bertino, "RWGuard: A real-time detection system against cryptographic ransomware," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 114–136, 2018.

[13] J. Gómez-Hernández, L. Álvarez-González, and P. García-Teodoro, "R-Locker: Thwarting ransomware action through a honeyfile-based approach," *Computers & Security*, vol. 73, pp. 389–398, 2018.

[14] D. Morato, E. Berrueta, E. Magaña, and M. Izal, "Ransomware early detection by the analysis of file sharing traffic," *Journal of Network and Computer Applications*, vol. 124, pp. 14–32, 2018.

[15] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A large-scale, automated approach to detecting ransomware," in *25th USENIX Security Symposium (USENIX Security 16)*, pp. 757–772, 2016.

[16] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, "Cryptolock (and drop it): stopping ransomware attacks on user data," in *36th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 303–312, 2016.

[17] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, "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.

[18] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "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.

[19] Q. Chen and R. A. Bridges, "Automated behavioral analysis of malware: A case study of wannacry ransomware," in *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 454–460, 2017.

[20] L. Zhao and M. Mannan, "TEE-aided write protection against privileged data tampering," *arXiv preprint arXiv:1905.10723*, 2019.

[21] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "A 0-day aware crypto-ransomware early behavioral detection framework," in *International Conference of Reliable Information and Communication Technology*, pp. 758–766, 2017.

[22] B. Lelonek and N. Rogers, *Make ETW greate again*. https://ruxcon.org.au/assets/2016/slides/ETW_16_RUXCON_NJR_no_notes.pdf.

[23] *What systems have you seen infected by ransomware?* https://www.statista.com/statistics/701020/major-operating-systems-targeted-by-ransomware/.

[24] *About Event Tracing*. https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing.

[25] *Krabsetw*. https://github.com/microsoft/krabsetw.

[26] A. Bayaga, "Multinomial logistic regression: Usage and application in risk analysis.," *Journal of applied quantitative methods*, vol. 5, no. 2, 2010.

[27] *VirusTotal*. https://www.virustotal.com/.

[28] *A Live Malware Repository*. https://github.com/ytisf/theZoo.

[29] *Malware samples*. https://github.com/fabrimagic72/malware-samples.

[30] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis, "Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts," in *IEEE Symposium on Security and Privacy (SP)*, pp. 1009–1024, 2017.

[31] *VirtualBox*. https://www.virtualbox.org.

[32] D. Nieuwenhuizen, "A behavioural-based approach to ransomware detection," *Whitepaper. MWR Labs Whitepaper*, 2017.

[33] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "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.

[34] M. M. Ahmadian, H. R. Shahriari, and S. M. Ghaffarian, "Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares," in *IEEE International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)*, pp. 79–84, 2015.

[35] M. M. Hasan and M. M. Rahman, "RansHunt: A support vector machines based ransomware analysis framework with integrated feature set," in *20th IEEE International Conference of Computer and Information Technology (ICCIT)*, pp. 1–7, 2017.

[36] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen, "Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, pp. 90–102, 2014.

[37] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, "Paybreak: Defense against cryptographic ransomware," in *Proceedings ACM on Asia Conference on Computer and Communications Security*, pp. 599–611, 2017.

[38] M. Hirano and R. Kobayashi, "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.

[39] M. Musch, C. Wressnegger, M. Johns, and K. Rieck, "Web-based cryptojacking in the wild," *arXiv preprint arXiv:1808.09474*, 2018.

APPENDIX A
BENIGN APPLICATIONS RESEMBLING CRYPTO RANSOMWARE IN BEHAVIOR

TABLE XVI: Benign applications generating a large number of file I/O events for compression, encryption or shredder.

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	-

APPENDIX B
BENIGN APPLICATIONS SPAWNING A LARGE NUMBER OF CHILD PROCESSES

TABLE XVII: Benign applications that may create process tree resembling locker ransomware.

Type	Application	Version	Locker-like?
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	-	×