Neural Computing and Applications (2022) 34:12077–12096
https://doi.org/10.1007/s00521-022-07096-6

ORIGINAL ARTICLE

Evading behavioral classifiers: a comprehensive analysis on evading ransomware detection techniques

Fabio De Gaspari[1] · Dorjan Hitaj[1] · Giulio Pagnotta[1] · Lorenzo De Carli[2] · Luigi V. Mancini[1]

Received: 31 July 2021 / Accepted: 14 February 2022 / Published online: 16 March 2022

Abstract
Recent progress in machine learning has led to promising results in behavioral malware detection. Behavioral modeling identifies malicious processes via features derived by their runtime behavior. Behavioral features hold great promise as they are intrinsically related to the functioning of each malware, and are therefore considered difficult to evade. Indeed, while a significant amount of results exists on evasion of static malware features, evasion of dynamic features has seen limited work. This paper examines the robustness of behavioral ransomware detectors to evasion and proposes multiple novel techniques to evade them. Ransomware behavior differs significantly from that of benign processes, making it an ideal best case for behavioral detectors, and a difficult candidate for evasion. We identify and propose a set of novel attacks that distribute the overall malware workload across a small set of independent, cooperating processes in order to avoid the generation of significant behavioral features. Our most effective attack decreases the accuracy of a state-of-the-art classifier from 98.6 to 0% using only 18 cooperating processes. Furthermore, we show our attacks to be effective against commercial ransomware detectors in a black-box setting. Finally, we evaluate a detector designed to identify our most effective attack, as well as discuss potential directions to mitigate our most advanced attack.

Keywords Ransomware · Machine learning · Behavioral detection · Evasion

1 Introduction

The problem of automatic malware detection is a difficult one, with no full solution in sight despite decades of research. The traditional approach—based on analysis of static signatures of the malware binary (e.g., hashes)—is increasingly rendered ineffective by polymorphism and the widespread availability of program obfuscation tools [1, 2]. Using such tools, malware creators can quickly generate thousands of binary variants of functionally identical samples, effectively circumventing signature-based approaches.

As a result, in recent years the focus of the community has increasingly shifted toward dynamic, behavior-based analysis techniques. Behavioral approaches sidestep the challenges of obfuscated binary analysis. Instead, they focus on the runtime behavior of malware processes, which is difficult to alter without breaking core functionality, and is therefore considered a reliable fingerprint for malware presence. This strong push toward malware behavioral analysis, coupled with the recent improvements in the field of machine learning (ML), has resulted in a multitude of ML-based behavioral approaches to malware detection. Popular techniques range from modeling of system call sequences [3] to full-fledged, fine-grained modeling of process behavior [4, 5]. At first sight, these techniques seem to hold great promise: the behavior of malware

✉ Fabio De Gaspari
degaspari@di.uniroma1.it

Dorjan Hitaj
hitaj.d@di.uniroma1.it

Giulio Pagnotta
pagnotta@di.uniroma1.it

Lorenzo De Carli
ldecarli@wpi.edu

Luigi V. Mancini
mancini@di.uniroma1.it

[1] Dipartimento di Informatica, Sapienza Università di Roma, Rome, Italy

[2] Department of Computer Science, Worcester Polytechnic Institute, Worcester, United States

differs significantly from that of benign processes and ML-based behavioral models can easily and reliably exploit this difference to distinguish between these two classes of processes. Moreover, behavioral-based approaches are also able to correctly detect unseen malware samples, as long as these new samples exhibit some form of anomalous behavior with respect to benign processes, as showed by several recent works [4–8]. Finally, behavioral detectors generally use malware features arising from operations that are required to achieve the desired malicious behavior, and therefore are extremely hard to disguise or evade.

This work proposes a new set of techniques to evade behavioral features used by classifiers to detect malware. Using these techniques, we show that it is possible to evade features that are typically considered hard to disguise and that are inextricably linked to the behavior of malware processes. Furthermore, we thoroughly assess the robustness of recently-proposed behavioral-based ransomware detection models against our proposed attacks [4, 5]. We use ransomware as case study due to both the gravity of the threat (e.g., [9, 10]), and the fact that—given its highly distinctive behavioral profile—ransomware is a nearly ideal target for behavioral-based detection. Our results show that it is possible to craft ransomware that accomplishes the goal of encrypting all user files, and at the same time avoids generating any significant behavioral features.

Our proposed attacks have fairly low implementation complexity, do not limit ransomware functionality in any significant way, and were found to be effective against a set of academic and commercial anti-ransomware solutions. Moreover, our attacks are successful even in a black-box setting, with no prior knowledge of the tools' inner workings, their training data, or the features used by the ML model. The core of our approach is an algorithm that cleverly distributes the desired set of malware operations across a small set of cooperating processes. While our work has focused on obfuscating ransomware-related features, the underlying principles are general and likely to apply to a wide range of behavioral detectors that analyze the runtime behavior of different types of malware. To the best of our knowledge, this is the first instantiation of an efficient, practical collusion attack in the domain of ransomware. Finally, we complement our investigation of attacks with a discussion of potential countermeasures, as well as the design and evaluation of a detector designed to identify our most effective attack.

In this paper, which is an extended version of our previous conference paper [11], we make the following contributions:

- We perform a comprehensive analysis of characteristic features typically used to detect ransomware, and define techniques and criteria for evasion.
- We assess the robustness of current state-of-the-art behavioral ransomware detectors, showing how it is possible to design ransomware that completely evades detection. In particular, we analyze three evasion techniques: *process splitting*, *functional splitting*, and *mimicry*.
- We implement and evaluate Cerberus, a proof-of-concept prototype of a ransomware following our approach, proving that our evasion techniques are practical.
- We evaluate our novel evasion techniques against multiple state-of-the-art ML detectors, as well as against a leading commercial behavioral detector. Results show that our techniques are effective and successfully evade detection, even in a black-box setting.
- We evaluate the dependence of our attack on the dataset used. Results show that our evasion techniques are effective even without access to the dataset used to train the target classifiers.
- We implement and evaluate a detector for our most effective attack, functional splitting, showing that it is possible to train a classifier to accurately detect this type of attack.
- We study if and how well the functional splitting detector generalizes on unseen functional-split ransomware. Our results show that the classifier is indeed robust and can generalize, motivating the need for more complex evasion attacks such as our proposed mimicry attack.

The remainder of the paper is structured as follows: Sect. 2 provides background on adversarial ML and ransomware detection. Section 3 describes our novel evasion techniques. Section 4 analyzes features used for ransomware detection, and discusses general principles for evading detection. Section 5 describes a proof-of-concept ransomware implementing our approach, while Sect. 6 evaluates it against a suite of state-of-the-art detection techniques. Section 7 discusses possible countermeasures (beyond the ones evaluated in Sect. 7). Section 8 reviews related work, Sect. 9 discusses the ethical implications of our work, and Sect. 10 concludes the paper.

2 Background

2.1 Adversarial ML

The core problem of adversarial ML can be stated in a simplified form as follows. Consider a multi-dimensional feature space V, where a vector $v \in V$ may represent properties (features) of an underlying object of interest o.

For example, a process may be represented by a vector encoding the frequencies of certain system calls. Furthermore, consider a classification function $f : V \rightarrow C$ mapping vectors in V to classes in a given set $C = c_1, \ldots, c_n$. Typically the goal of an adversarial ML attack is to cause f to misclassify one or more feature vectors, i.e., forcing the classifier to make mistakes in mapping objects to labels. Many classifiers used in security distinguish between a benign and a malicious object class, i.e., $C = \{B, M\}$. In this setting, the problem therefore becomes—given a vector v belonging to class M—to cause $f(v) = B$. One way to "trick" f is to alter its training dataset, causing it to learn an incorrect boundary between classes. This constitutes a poisoning attack [12]. Conversely, *adversarial sample generation* consists in picking a victim classifier which has been trained on a correctly-labeled dataset, and crafting one or more malicious objects in such a way that they get classified as benign. In our work, we focus on the latter attack.

The literature proposes several techniques to generate adversarial feature vectors directly from a formal representation of a target classifier (e.g., [13, 14]). These techniques typically are agnostic to the type of object being studied, and work purely in feature space. There is then a second line of work which investigates the complementary problem, given a successful adversarial feature vector, to generate a corresponding object (e.g., given the feature vector that a stealth malware should exhibit not to be detected, generate the actual binary of the malware). Past work demonstrates use of adversarial ML to generate stealth PDF exploits [15–17], Android malware [18–20], Flash malware [21], and a variety of other dangerous objects. Our work falls within the realm of adversarial sample generation. In the rest of this paper, we demonstrate (1) heuristics to generate adversarial feature vectors for ransomware behavior, and (2) a proof-of-concept ransomware prototype whose behavior generates the target adversarial feature values.

2.2 Behavioral ransomware detection

The literature presents several recent works on ransomware detection based on behavioral features [4–7, 22]. UNVEIL [6] and its successor Redemption [7] detect suspicious activity by computing a score using an heuristic function over various behavioral features: file entropy changes, writes that cover extended portions of a file, file deletion, processes writing to a large number of user files, processes writing to files of different types, back-to-back writes. Similarly, CryptoDrop [22] maintains a "reputation score"—indicating the trustworthiness of a process—computed based on three main indicators: file type changes, similarity between original and written content, and entropy measurement. Additionally, CryptoDrop also uses deletion and file type funneling (reading/writing a small set of file types) as secondary indicators.

Our attack is motivated by a review of all the approaches cited above; for evaluation however we selected two of them, which are described in greater detail in the following. The selection was based on practical considerations: both approaches were published in highly visible venues, and in both cases the authors kindly provided enough material (code and/or datasets) and support to enable us to run their software. Our evaluation also includes a commercial product from Malwarebytes (discussed at the end of this section).

2.2.1 ShieldFS

ShieldFS [4] is a technique for identifying ransomware processes at file-system level and transparently roll back file changes performed by processes deemed malicious. Ransomware detection is based on ML models of well- and ill-behaved processes. Detection is performed at the process level by using a hierarchy of random forest classifiers tuned at different temporal resolutions. Using different temporal resolutions allows ShieldFS to take into account both short- and long-term process history when performing classification, which is crucial to detect code injection-based ransomware. ShieldFS uses features typically associated with ransomware operation for the classifier, such as #folder-listing operations, #read operations, #write operations, #rename operations, percentage of file accessed among all those with same extension and average entropy of data passed to write operations.

ShieldFS divides the lifetime of each process in up to 28 *ticks*; ticks do not represent fixed interval of times; instead, they define fractions of the overall set of files accessed by a process. Ticks are exponentially spaced; the first tick is reached when a process has accessed 0.1% of the files on the filesystem; the last when a process has accessed 100% of the files. Whenever a certain tick i is reached, ShieldFS computes the features over multiple intervals. The first interval covers operations between ticks $i - 1$ and i. Each of the remaining intervals ends at tick i and begins further in the past compared to the previous one. Features computed over each interval are fed to a dedicated model for classification. Figure 1 (reproduced from [4]) shows the

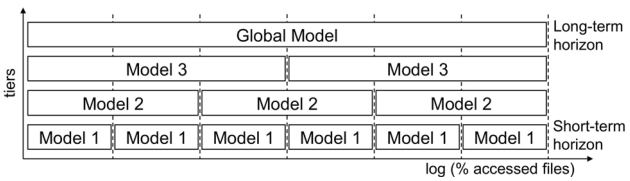


Fig. 1 Incremental models in ShieldFS (reproduced from [4])

first six ticks in the lifetime of a process, and the various intervals covered by each model. A process is considered malicious if positively detected for $K = 3$ consecutive ticks.

ShieldFS also employs a system-wide classifier that computes and classifies feature values across all processes in the system. This classifier is however only used to disambiguate ambiguous results from per-process classifiers. In other words, if a per-process model over a certain interval cannot determine whether the process is malicious or not, system-wide features are computed over the same interval and fed to the system-wide classifier. When our attack is successful, individual processes are always classified as benign with high confidence, and therefore the system-wide classifier is never triggered.

2.2.2 RWGuard

RWGuard [5], by Mehnaz et al., is a ransomware detector which leverages multiple techniques: process behavior, suspicious file changes, use of OS encryption libraries, and changes to decoy files. We do not discuss decoy and library-based detection as it is orthogonal to our work. Compared to ShieldFS, RWGuard uses a relatively simple detector consisting of a random forest classifier that analyzes process behavior using a 3 s sliding window. The features used by the classifier include the number of various low-level disk operations performed by each process under analysis. The behavioral classifier is complemented by a file monitor component which computes four metrics after each write operation: a similarity score based on similarity-preserving hashing, size difference, file type change, and file entropy. Significant changes in any of first three metrics and/or high file entropy are interpreted as a sign of ransomware activity.

The detection process of RWGuard consists of three steps: when the behavioral classifier detects a suspicious process activity, the file monitor component is invoked to validate the initial detection. If both modules agree that the activity is suspicious, a third module, the File Classification module, is invoked to assess if the encryption operation is benign or malicious. Only after all three modules agree on the maliciousness of the suspicion activity, then the responsible process is considered malicious. When our attack is successful, individual processes are classified as benign by the behavioral module, and the remaining modules are not invoked.

2.2.3 Malwarebytes

Several commercial anti-ransomware solutions exist; for our work, we chose to evaluate Malwarebytes' Anti-Ransomware [23]. Differently from most other vendors, Malwarebytes distributes the beta versions of their ransomware detector as a discrete component, i.e., one which is not integrated with other types of anti-virus technology. This enables us to evaluate ransomware detection performance without having to account for interference from other malware/virus detection modules. Malwarebytes does not provide details on the inner workings of their product; the company however states that their product "does not rely on signatures or heuristics" [23] and leverages machine learning [24]. These indications suggest some type of behavioral classifier. For our evaluation, we use version 0.9.18.807 beta.

3 Evading behavioral detectors

Behavioral classifiers are designed to use features that are considered inextricably linked with malicious behavior and generally not present in benign applications. Our approach is based on the insight that behavioral detectors collect these features on a per-process basis to model the behavior of a given application. For instance, ransomware detectors profile processes based on features such as entropy of write operations or number of read/write/directory listing operations. We exploit this limitation by devising a novel evasion technique based on distributing the malware operations across multiple independent processes: each process individually appears to have a benign behavior. However, the aggregated actions of all these processes result in the intended malicious malware behavior. It is important to note that this is not just a limitation of current behavioral classifiers, but it is rather an inherent restriction of process behavioral modeling, as there is no straightforward way to identify a set of processes working independently to achieve a common final goal [25]. While communication among coordinating processes could be used to infer cooperation, such communication can be limited and/or hidden using covert channels. Moreover, it is possible to employ techniques to avoid hierarchical relationships between processes (e.g., parent-child) [25]. Section 5 discusses inter-process communication in our prototype and explains how it can be made completely stealthy.

The remainder of this section describes our three proposed approaches to evade classification, in increasing order of complexity: process splitting, functional splitting, and mimicry.

3.1 Process splitting

is the simplest and most straightforward multi-process ransomware evasion approach. In process splitting (depicted in Fig. 2b), the ransomware behavior is distributed

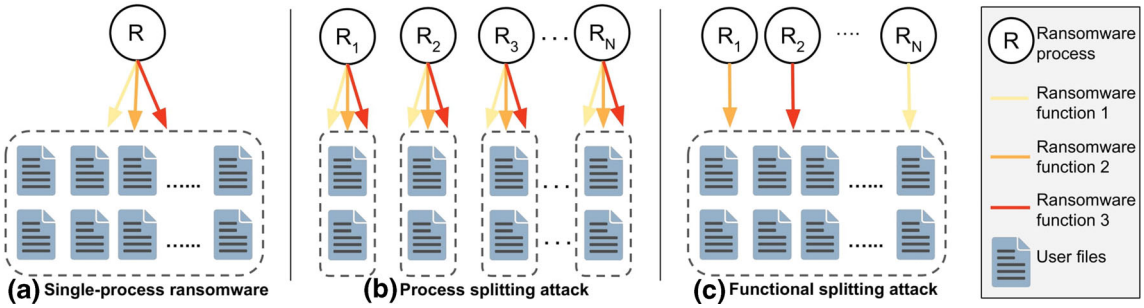


Fig. 2 Process and functional splitting attacks

over N processes, each performing $1/N$ of the total ransomware operations. Effectively, this approach implements a form of data parallelism: each individual process performs all the ransomware operations on a subset of the user files. The intuition is that ransomware classifiers are trained on traditional, single-process ransomware, which exhibits extremely high number of operations such as directory listing, read, and write. Splitting the ransomware over N independent processes allows to reduce the number of such operations performed by each individual processes, since each process only encrypts a subset of all files. If we split the original ransomware enough times, the number of operations performed by each individual *process-split ransomware process* becomes low enough that the classifier is unable to detect the ransomware.

While this technique is simple, our experiments show it can be extremely effective even against complex classifiers (see Sect. 6). Moreover, the approach can be easily tailored against specific ML models by following a simple process splitting procedure. Given a trained classifier we want to evade, we run our ransomware and query the model to check if it is detected. If the ransomware is detected, we further split its operations and query the model again. We continue splitting the ransomware operations over an increasing number of processes and querying the classifier until the desired evasion rate is achieved.

3.2 Functional splitting

While process splitting is very effective in reducing the accuracy of ransomware classifiers, completely evading detection can be challenging. Indeed, depending on the robustness of the target classifier, process splitting might require creating a very large number of processes, which in turn could be used to detect the presence of ransomware. A more well-rounded approach to classifier evasion is *Functional Splitting*, as illustrated in Fig. 2c. Ransomware processes perform a set of operations (or *functions*) to encrypt user files, such as reading, writing, or directory listing. When using functional splitting, we separate each of these ransomware functions in a process group: each process within the group (called *functional split ransomware process*) performs only that specific ransomware function. Within each group, we can further apply process splitting to reduce the number of operations performed by an individual functional split ransomware process. The intuition behind the functional splitting approach is that ML classifiers use groups of features to classify processes. If a process only exhibits a small subset of the features that the model associates to ransomware, then it will not be classified as malicious. Functional splitting takes this concept to the extreme, having each functional split ransomware process only exhibits a single ransomware feature.

3.3 Mimicry

Functional splitting is extremely effective against current state-of-the-art ransomware classifiers. Moreover, it does not suffer from the process explosion issue that affects process splitting. However, it could be feasible to train an ML model to recognize this particular evasion attack. Typical benign processes perform several different types of functions, therefore an ML model could be trained to differentiate between benign processes and functional split ransomware processes. Indeed, our evaluation shows that this is in fact possible. In Sect. 6.5, we evaluate a classifier trained on both normal ransomware as well as on functional split ransomware. Our evaluation shows that such a classifier is able to detect functional split ransomware with high accuracy. Furthermore, we show that a classifier trained on a functional split ransomware with a low number of inner splits (i.e., each process in a functional group is split only few times) can generalize and correctly classify functional split ransomware with high number of inner splits.

In order to fully evade behavioral ransomware classifiers and avoid the drawbacks of functional splitting, we propose a third evasion attack: *Mimicry*. Rather than splitting ransomware processes into individual functional groups, each ransomware process is designed to have the same functional behavior as a benign process, effectively making it

indistinguishable from other benign applications. The intuition behind the mimicry approach is that behavioral ML models classify samples based on the expression of a given set of features. Ransomware processes exhibit some characteristic features, while different benign applications exhibit different sets of features to different degrees. By splitting ransomware into multiple processes—and having each individual process exhibit only features displayed by benign processes—it becomes impossible for a classifier to distinguish between the runtime behavior of *mimicry ransomware processes* and benign processes. Effectively, mimicry ransomware processes are modeled after benign processes and exhibit only features that benign processes exhibit. Moreover, the degree to which each feature is exhibited by each mimicry process (e.g., how many read/write operations are performed) is kept consistent with that of benign processes.

The end result of the mimicry approach is ransomware processes that act exactly like benign processes. However, the collective behavior of all the mimicry processes results in the desired malicious end goal. Section 4 discusses which features are characteristic of ransomware processes, and how we can limit the occurrence of each of these features in order to mimic the behavior of benign processes.

4 Features discussion

Behavioral classifiers exploit the marked behavioral differences between benign programs and malware in order to detect malicious samples. In the context of ransomware, such classifiers rely on a wide range of features that all ransomware programs must exhibit in order to reach their goal. This section discusses these features and analyze their robustness to evasion. Many of the features described here are also displayed by benign processes, and each feature by itself does not provide strong evidence for or against ransomware behavior. However, when considered together, these features highlight a very unique program behavior proper of ransomware processes.

4.1 Write entropy

The end goal of ransomware is to encrypt users' files and collect a ransom payment in exchange for the decryption key. Typical encrypted data are a pseudorandom string with no structure, and exhibit maximum entropy [5], while structured data written by benign programs are assumed to have considerably lower entropy. Consequently, entropy of write operations appears to be a useful feature to differentiate ransomware from benign processes. In fact all state-of-the-art ML ransomware detectors use entropy of write operations as a feature, in one form or another [4–7].

Evasion Entropy as a feature for ransomware detection can be used at different levels of granularity: (1) overall file entropy [5], (2) average read-write operations difference [6, 7], and (3) individual write operations [4]. Feature (1) does not allow accurate differentiation between ransomware and benign processes, as nowadays most common file types are compressed for efficiency, including file types generally targeted by ransomware such as pdf, docx, xlsx, video, and image files. Consequently, the overall file entropy for this file types, as measured by current state-of-the-art approaches relying on Shannon entropy [26], is comparable to that of an encrypted file. For what concerns feature (2), in our research we analyzed several file types with their associated programs, and found out that in general benign processes working on compressed formats exhibit numerous very high entropy reads and writes.

It is worth pointing out, however, that despite the considerations above our dataset also shows a non-negligible difference in the average entropy of individual file *write operations*. Such averages are 0.4825 for benign processes vs 0.88 for ransomware, with range [0–1]. Despite this somewhat counter-intuitive result, it is still straightforward to evade feature (3). Average write entropy can be skewed simply by introducing artificial, low-entropy write operations that lower the average write entropy for a ransomware process, bringing it in line with that of benign processes.

4.2 File overwrite

Different ransomware families use different techniques to encrypt user files. However, one feature that is common across ransomware families is that the original file is fully overwritten, either with the encrypted data or with random data to perform a secure delete [6]. On the other hand, benign processes rarely overwrite user files completely. Therefore, file overwrite is a valuable feature that can be exploited to classify between ransomware and benign processes.

Evasion In order to evade this feature, the percentage of a file overwritten by a single ransomware process must be limited. Maintaining this percentage within the range exhibited by benign processes can be easily achieved with our proposed multi-process ransomware approach. It is sufficient to distribute write operations to a given file over multiple ransomware processes, each of which only overwrites a portion of the file. Each individual process does not show any suspicious behavior, but the aggregated action of all the processes still overwrites the whole file.

4.3 Directory traversal

In order to maximize the amount of damage for the victim, ransomware typically encrypts every file in any given user directory. Therefore, ransomware processes issue *open* operations for every file in a directory. This behavior is fairly distinctive and not common in benign processes, except for some particular cases. Therefore, when coupled with other meaningful features, directory traversal can serve as a useful indicator to detect a ransomware process.

Evasion The directory traversal feature can be easily evaded using our multi-process ransomware approach. Indeed, by setting an upper bound on the number of files each ransomware process can access in a given directory, it is possible to heavily reduce the expression of this feature.

4.4 Directory listing

In order to encrypt user files, ransomware first needs to discover all files of interest. In order to perform file discovery, ransomware issues a very large amount of file listing operations. While some types of benign processes also exhibit similar behavior (e.g., Windows Explorer), this feature can be a valuable indicator of ransomware activity when coupled with other meaningful features.

Evasion The directory listing feature can be easily evaded using our multi-process ransomware approach. Indeed, having multiple processes allows to distribute the listing operations and therefore limit the incidence of this feature for each individual process.

4.5 Cross-file type access

Typically, benign processes only access a fixed subset of file types (e.g., a pdf viewer will access pdf files, but not docx). On the other hand, a ransomware process will access all important user files, regardless of their file types, in order to encrypt them (e.g., pdf, xlsx, jpeg). This cross-file type access behavior can therefore serve as a useful feature to detect ransomware operations when coupled with other meaningful features.

Evasion Cross-file type access can be evaded in a similar fashion to the directory traversal feature. It is sufficient to separate ransomware processes into different groups, and have each process in a group access only a coherent subset of file types. This is enough to make a ransomware process indistinguishable from benign processes from the point of view of file type access.

4.6 Read/write/open/create/close operations

Ransomware accesses and encrypts as many files as possible on the victim's directories to maximize the damage and ensure the payment of a ransom. This behavior results in an abnormally large amount of file operations such as *read, write, open, close* and, for some ransomware families, *create*. Typical benign processes rarely access so many files in a single run, except for some particular cases (e.g., files indexer).

Evasion Our multi-process approach allows to evade this feature. By using multiple coordinated processes to encrypt user files, each individual process only needs to access a subset of all user files. By varying the number of ransomware processes used, we can limit how many file operations each individual ransomware process performs.

4.7 Temporary files

Some ransomware families use temporary files as buffer to encrypt files before overwriting them, or as a temporary data storage while copying or removing the original files [5]. While several benign programs also generate temporary files, this behavior can be useful when used together with other indicators to detect the presence of ransomware [5].

Evasion Ransomware does not necessarily need to use temporary files to encrypt user data. Therefore, the most obvious and effective way of evading this feature is not creating temporary files. Regardless, our multi-process technique can be applied to keep the number of temporary files generated by each individual process, in line with the average number of temporary files used by benign applications.

4.8 File type coverage

For the purpose of ransomware, not all files are equally important. Some file types are far more likely to contain important data for the victim than others (e.g., a docx file vs. a cfg). Ransomware strives to access and encrypt all files with relevant extensions [4], such as all docx files in a directory tree, in order to maximize damage. On the other hand, benign programs typically only access a fraction of these files.

Evasion While it can be tricky for traditional ransomware to evade this feature, it is fairly straightforward when using our proposed approach. A natural consequence of distributing ransomware over multiple processes is that each individual process accesses only a fraction of the total user files. To evade the file type coverage feature, it is sufficient

to make sure that each ransomware process only accesses and encrypts a portion of all the files of a given type.

4.9 File similarity

Encrypting a file completely changes its content, since the original data are overwritten with pseudorandom data. Typical benign processes rarely alter a file in a way that its content is overall completely different from the previous version. On the other hand, ransomware always completely changes the whole content of a file when encrypting it. Therefore, overall file similarity before and after write operations from a given process is a strong feature to detect ransomware operation [22].

Evasion Traditional ransomware always changes the whole file, either by encrypting it or by performing a secure delete operation. Therefore, there does not appear to be a simple way to avoid file similarity feature. However, our approach can evade this feature with a technique similar to that proposed for the file overwrite feature. By having each ransomware process encrypt only a portion of any given user file, we preserve the overall file similarity after each individual write operation, and no individual process changes the whole file content.

4.10 File-type change

Files are structured data, and most file types are characterized by some *magic bytes* [27] within the file content. When a file is fully encrypted, the magic bytes are also altered, effectively changing the file type signature. Benign program operations do not generally alter file types. On the other hand, ransomware operations always do. Therefore, file type change can be a useful feature to detect ransomware operation.

Evasion The file type change feature can be easily evaded, even by traditional ransomware. Indeed, it is enough for the ransomware to preserve the original magic bytes of the file. Even if more complex file-type techniques are used, such as looking for changes in the overall structure of a given file, a ransomware can still evade this feature by maintaining the overall file structure during encryption with techniques such as format-preserving encryption [28].

4.11 Access frequency

In order to maximize the damage to the victim and minimize the time window to intervene and stop the attack, ransomware aims to encrypt user files as quickly as possible. To do so, a typical ransomware performs write operations on different files in short time windows. Some ransomware classifiers therefore use the write access frequency of a process as a feature to differentiate between benign and malicious processes [7].

Evasion Our analysis indicates that even benign processes often exhibit high file-write access frequency. Analyzing the ShieldFS process dataset [4], we found that it is not uncommon for benign processes to perform write operations to different files in short time intervals, making this feature rather weak for ransomware classification. Moreover, using our multi-process approach allows to decrease the ransomware access frequency to be in line with that of benign processes, while maintaining a high file encryption throughput due to parallelization.

4.12 Other features

This section covered what we found to be the most used and robust features employed by current ransomware classifiers. Other features were proposed in the literature to improve detection accuracy. However, we found that these are either very weak (e.g., file size change [5]), or extremely similar to other features that we already discussed (e.g., file type funneling [22]). Therefore, evasion is either not necessary or achieved in a similar way to what is discussed.

5 Implementation

This section presents the implementation of Cerberus, our ransomware prototype implementing the mimicry evasion technique, as well as our re-implementation of the ShieldFS classifier.

5.1 The cerberus prototype

Splitting ransomware functionality over independent processes requires coordination. Ensuring that each ransomware process only expresses features typical of a benign process further complicates the implementation. Here, we briefly describe Cerberus, a new ransomware prototype developed to demonstrate the feasibility of our evasion techniques. The Cerberus prototype implements both the *functional splitting* and *mimicry* attacks. Functional splitting separates ransomware functions in different *functional groups*: a process in any given group performs only the specific ransomware functions assigned to that group. For instance, ransomware processes in the read-write functional group only perform read and write operations. Cerberus implements functional splitting by separating ransomware operations in three groups: (1) directory list, (2) write, and (3) read-rename. Read and rename are performed in the same functional group mainly for

implementation convenience. Note that separating read and write does not require additional open operations, as Cerberus uses the Windows API `DuplicateHandle()` to share handles to opened file between collaborating ransomware processes. We could have considered additional features for the implementation of functional splitting, as discussed in Sect. 4. However, since the goal of Cerberus is merely to prove the feasibility of our evasion techniques, we considered only the most important features exhibited by every ransomware family. Section 6.3 shows that the features considered are enough to evade even commercial ransomware detectors in a black-box settings.

To implement the mimicry attack in Cerberus, we performed a statistical analysis on the behavior of benign processes from the ShieldFS dataset (Sect. 6.1), which contains traces from 2245 unique benign applications collected over a month. Table 2 shows that we can identify a few behavioral classes that represent most benign processes in the dataset (notation in Table 1). For our implementation, we chose the 2nd and 3rd most represented classes: directory listing-read and read-write-rename. We chose these because they are highly represented in the dataset of benign processes, as well as because no ransomware process belongs to any of these two classes (all the 383 real ransomware in our dataset exhibits directory listing, read, write, and rename operations together). Within each class, we strive to maintain the same ratio between operations as exhibited by benign processes. To achieve this, we introduce dummy operations, such as null reads or empty writes, to maintain the exact operation ratio of benign processes. As creating a large number of processes at the same time could be used to detect our evasion technique, Cerberus' ransomware processes are generated in a sequential fashion. It is worth noting that this is not required, and that in general few ransomware processes can be generated at a time in order to improve throughput.

It is worth noting that, while our Cerberus prototype works under Windows systems, an equivalent malware could be easily implemented in Linux and MacOS in a similar fashion.

Table 1 Notation

DL: Directory listing operation	CL: Close operation
RD: Read operation	FRD: Fast read operation
WT: Write operation	FWT: Fast write operation
RN: Rename operation	FOP: Fast open operation
OP: Open operation	FCL: Fast close operation
{*X*,*Y*}: Functional group of processes performing op. *X* and *Y*	

5.2 Ransomware interprocess communication

In order to orchestrate individual processes and achieve the ransomware's goal, it is necessary to coordinate them properly, which requires some form of inter-process communication. For instance, writer processes need to know the original content of the file to encrypt, which is provided by reader processes. However, standard inter-process communication mechanisms such as pipes generate additional I/O Request Packets (IRP) [29], which are used by current state-of-the-art detectors to calculate features. Therefore, using standard inter-process communication would skew the behavior of the mimicry ransomware processes, potentially making them easier to detect. The Cerberus prototype implements a stealthy communication technique based on direct memory access, which allows us to avoid generating additional IRP traces for process communication. In particular, we leverage a feature of the Windows API that allows processes belonging to the same user to read/write directly to each other's address space, without the need for any special permissions or memory sharing. This is possible since all ransomware processes are started by the same user. The only requirement is for the ransomware processes to know the address range from the address space of the process they wish to read/write to, which in our case is passed as an argument during process creation. This is mostly for implementation convenience, as we could easily use more sophisticated covert channel techniques to share the memory address in a stealthy manner.

A keen observer could make the point that direct memory access is effective only because current state-of-the-art detectors use IRP traces to build feature vectors of processes and that an improved detector could, for instance, use lower level system call hooks to intercept direct memory communication. However, multiple works have shown that detecting communication between several malware processes can be challenging, as many communication and covert channel techniques can be employed to thwart detection [25, 30] (these works are also discussed in Sect. 8). While there exist works aimed at detecting memory-based and other types of covert channel communication [31], the general problem of detecting any covert channel is still open. Cerberus is merely an experimental prototype and we did not implement more complex forms of stealth communication, however, this could be easily done in a real-world implementation of our attacks.

5.3 ShieldFS

As we could not obtain the original code or a prototype for ShieldFS due to patenting issues, we re-implemented the

ShieldFS classifier exactly as described, interacting with the ShieldFS's authors to clarify any potential misunderstanding. We split the ShiedFS dataset in training and testing set following a 10 : 1 ratio and trained each of the ShieldFS model's tiers with the appropriate feature vectors from benign and ransomware traces. As in the original paper, we trained multiple classifiers for each of the 28 tiers, covering percentage of file accessed from 0.1% up to 100%. Each classifier is implemented as a random forest of 100 trees. All parameters and details about the setup of the dataset and the ensemble of classifiers were set following the original setup in [4]. We validated our implementation on the training set and obtained results in line with the original classifier. More specifically we performed a one-machine-off cross validation, as done in the original paper, where benign data from one machine is selectively removed from the training set and used for the testing. The average accuracy of our classifier came out at 98.6%, which is slightly higher that the 97.7% overall performance reported in [4].

6 Evaluation

This section presents the experimental evaluation of our evasion techniques. In particular, we aim at answering the following research questions: (1) *is our theoretical attack technique effective in avoiding detection?* In Sect. 6.2, we apply our techniques to traces generated executing traditional ransomware, and show that process splitting, functional splitting, and mimicry effectively avoid detection; (2) *can our theoretical attack evade detection when implemented in a real-world setting?* In Sect. 6.3, we evaluate our prototype Cerberus in a virtual machine, showing that our theoretical attacks can be implemented and are effective in the real world; (3) *do our evasion techniques generalize, evading classifiers trained on different datasets?* In Sect. 6.3 we show that the mimicry attack, modeled on the ShieldFS dataset and implemented in Cerberus, successfully evades detection of RWGuard, which is trained on a different dataset; (4) *is our attack effective in a black-box setting against commercial behavioral ransomware detectors?* Sect. 6.4 shows that Cerberus successfully evades detection of Malwarebytes Anti-Ransomware, a leading commercial behavioral ransomware detection tool; (5) *can a behavioral-based classifier detect with high accuracy functional-split ransomware? Can such a detector generalize to unseen functional split ransomware?* In Sect. 6.5, we show that an appropriately designed and trained detector can detect functional-split ransomware with high accuracy and that such model can generalize to unseen functional-split ransomware.

6.1 Dataset and experimental setup

Our trace-based evaluation leverages a dataset provided to us by the authors of ShieldFS. Table 3 summarizes this dataset; further details can be found in [4]. To train our classifiers, we divided the data on benign processes from the 11 machines comprising the dataset into: 10 machines for the training set and one for the testing set. For the 383 ransomware samples, which include different ransomware families, we use 341 for training and 42 for testing.

In order to test our Cerberus prototype, we created a realistic virtual machine-based testbed, consisting of a VirtualBox-based Windows-10 VM. We based the VM user directory structure and file types on the disk image of an actual office user. File contents were extracted from our own machines and replicated as needed. In total, our VM is comprised of 33,625 files for a total of $\sim 10\,GB$, distributed over 150 folders.

In all our experiments we use only ransomware processes for the evaluation, since the goal is to assess whether our proposed approaches can successfully evade detectors. Consequently, for all our experiments we use the accuracy of each classifier in detecting ransomware processes as performance metric.

6.2 Trace-based evaluation

This section presents the trace-based evaluation of process splitting, functional splitting, and mimicry attacks. This evaluation uses the I/O Request Packets (IRP) traces [29] of real ransomware from the ShieldFS dataset. The testing dataset contains 42 unique ransomware samples, which include different ransomware families. For each ransomware, the IRP trace contains the complete list of I/O operations performed by the ransomware process. Both ShieldFS and RWGuard extract the ransomware features used for detection, such as number of read/write operations, directly from the IRP Traces.

Our evaluation simulates multiple processes by splitting the IRP trace of a single ransomware in multiple traces, based on the specific evasion technique under evaluation. Successively, we compute the feature vector for each individual trace as if it were an individual ransomware process. Finally, we query the classifier and compute the percentage of feature vectors classified as belonging to a ransomware. Table 1 introduces the notations that we will use in the remainder of this section.

6.2.1 ShieldFS

This section evaluates the effectiveness of our techniques against the ShieldFS ransomware detector.

6.2.1.1 Process splitting As mentioned in Sect. 3.1, process splitting evenly splits the operations performed by a ransomware process over N processes. In a process-split ransomware, all processes exhibit almost identical behavior and characteristics. We begin our evaluation by splitting the original ransomware trace in multiple traces, querying the classifier in each trace. We increase the number of traces until complete evasion is achieved. We evaluate process splitting with 42 unique ransomware traces, which include different ransomware families. We compute the feature vector for each process-split ransomware, query the classifier and compute the percentage of feature vectors flagged as malicious. Figure 3a illustrates our results. We can see that ShieldFS accuracy decreases already after a single split, going from single-process 98.6% accuracy down to 65.5% on a two-process ransomware. Further splitting incurs diminishing returns: going from two to ten processes results in 20.1% accuracy (45.4% decrease), and going from ten to one hundred processes results in 16.69% accuracy (3.41% decrease). Completely evading the ShieldFS classifier requires approximately 11000 processes. The requirement of such a large number of processes to achieve full evasion is a clear drawback of this simplistic approach. It is reasonable to imagine a countermeasure that can detect process-split ransomware by monitoring the process creation behavior at a system-level. Large swaths of newly created processes that exhibit similar behavior can then be clustered and analyzed as if they were a single process.

6.2.1.2 Functional splitting As discussed in Sect. 4, ransomware exhibits several distinctive features. Functional splitting (see Sect. 3.2) exploits the reliance of current behavioral classifiers on the presence of most of these features to detect ransomware. This section evaluates the effectiveness of evasion based on the lack of expression of certain features against ShieldFS. The ShieldFS classifier is trained on six features: #folder listing (DL), #file reads (RD), #file write (WT), #file rename (RN), file type coverage, and write entropy. Our evaluation focuses on the four main operations performed by ransomware—DL, RD, WT, RN—and split ransomware processes based on these four functional groups. Finally, we assess how each functional split ransomware process performs against the detector. Note that focusing only on these 4 features makes it harder to evade the detector, since we make no attempt to evade the remaining 2 features.j

Differently from process splitting, functional splitting requires the definition of the set of features that are targeted for evasion. In our experiments, we first evaluate single functional splitting, where each functional split process performs only one type of operation, resulting in four functional groups (DL, RD, WT, and RN process groups). Within each functional group, we apply our process splitting technique: starting with a single functional split process per group, we recursively split it into multiple processes until complete evasion is achieved. As illustrated in Fig. 4a, we are able to completely evade ShieldFS by using 20 functional split processes, 5 for each of the four functional groups. Note the contrast between Figs. 3a and 4a. With single functional splitting, 4 processes (one for each functional group) are enough to drop the detector accuracy down to $\sim$2.5%, compared to the $\sim$7500 processes required with process splitting.

The effectiveness of functional splitting can be explained by analyzing the dataset. There is a significant difference in behavior, in terms of types of operations performed, between benign and ransomware processes over their lifetime. All of the ransomware processes in the dataset perform DL, RD, WT, and RN types of operations, while only approximately 19% of benign processes have a similar behavior. Since the feature expression profile between traditional and functional split ransomware is so different, with the latter being closer to benign processes than traditional ransomware, the accuracy of the classifier is heavily affected. To validate this hypothesis, we further study how different functional groups affect the performance of the detector. In particular, using combined functional groups (i.e., processes performing RD and WT, or DL and RN), rather than single functional groups, should result in higher detection accuracy as the behavioral profile of the functional split ransomware gets closer to that

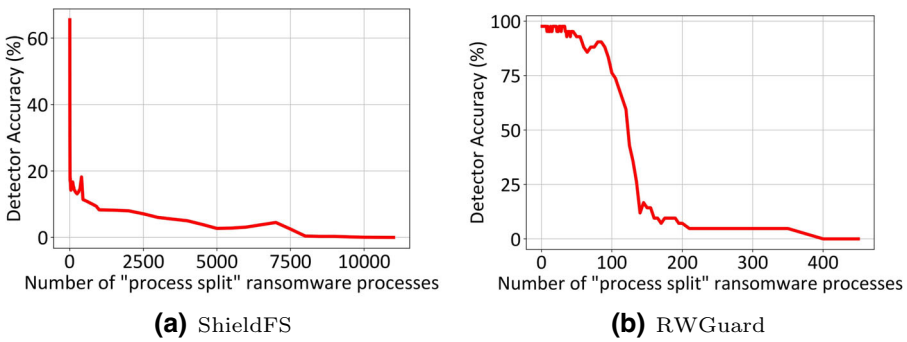
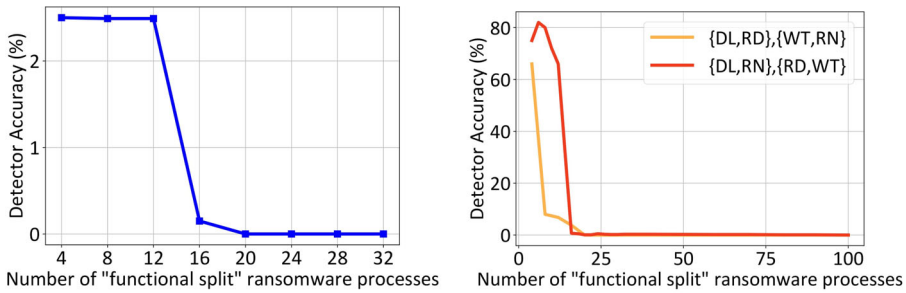


Fig. 3 Evaluation of the process splitting evasion technique



(a) Single functional splitting **(b)** Combined functional splitting

Fig. 4 Evaluation of the functional splitting evasion technique against ShieldFS

of a traditional ransomware. Figure 4b illustrates our results. This experiment evaluates the accuracy of ShieldFS considering two different implementations of functional split ransomware. In the first implementation, the operations are divided into the two functional groups {DL,RD},{WT,RN}, while in the second implementation the two functional groups are {DL,RN}, {RD,WT}. Figure 4b shows that the initial accuracy of the classifier is much higher when compared to single functional splitting, hovering around 80% for {DL,RN}, {RD,WT} and around 70% for {DL,RD},{WT,RN}. However, the accuracy quickly drops as we apply process splitting within each functional group, reaching $\sim$0% at 20 processes (10 for each functional group). The high initial detection accuracy for Fig. 4b is due to the fact that in the first ransomware implementation we have the {RD,WT} functional group and in the second implementation we have the {WT,RN} functional group. Both these functional groups are always present in traditional ransomware, therefore the model is more likely to classify processes that heavily exhibit these features as malicious. Indeed, we can see that after process splitting is applied in each functional group—and therefore the number of operations per functional split ransomware process decreases—the accuracy for both functional splitting implementations quickly falls toward zero.

6.2.1.3 Mimicry Functional splitting evades detection by preventing the expression of one or more features from ransomware processes. Mimicry takes this concept one step further: rather than completely removing a feature, we model ransomware features so that, on average, they are identical to those of benign processes. Our mimicry attack therefore requires the definition of a behavioral model (or set of behavioral models) of benign processes that the mimicry processes will follow. We build our model of a typical benign process by performing an in-depth statistical analysis on the behavior of benign processes in the ShieldFS dataset [4], which comprises observations of well above 1 month of data from 2245 unique benign applications and $\sim$1.7 billion IRPs. We compute the average value for the main features used to profile ransomware and we extract the ratios between different types of I/O operations performed by benign processes. Finally, we split the ransomware activity into multiple processes, based on average feature values and ratios. We evaluate our approach on the ShieldFS ransomware traces that are part of our testing set.

We focus on modeling the four main operations performed by ransomware and benign processes—DL, RD, WT, RN—together with the number of file accessed. Note that we could easily consider more features in our modeling, up to all features described in Sect. 4. However, since the goal of this evaluation is to prove the effectiveness of our techniques, it is sufficient to consider the most representative features. Table 2 shows the different behavioral profiles exhibited by benign process, along with how represented that behavior is in the dataset. As can be seen, the most represented functional group of benign processes exhibits all four main operations {DL,RD,WT,RN}, with

Table 2 Behavioral profiles exhibited by benign processes and their presence in the dataset

DL	RD	WT	RN	% of processes
✓	✓	✓	✓	19.07
✓	✓	–	–	18.37
–	✓	✓	✓	16.35
–	✓	–	–	11.44
✓	✓	✓	–	7.60
–	✓	–	✓	6.85
–	–	–	✓	6.21
–	✓	✓	–	5.61
✓	–	–	–	3.55
–	–	✓	✓	2.18
✓	✓	–	✓	1.76
–	–	✓	-	0.42
✓	–	–	✓	0.38
✓	–	✓	–	0.13
✓	–	✓	✓	0.08

Table 3 Dataset details

Type	Benign	Ransomware
# Unique applications	2245	383
# Applications training set	2074	341
# Applications testing set	171	42
# IRPs [Millions]	1763	663.6

the functional groups {DL,RD} and {RD,WT,RN} being a close second and third. On the other hand, if we consider the behavioral profile of ransomware processes, all 383 ransomware samples perform all four main operations. Given that the first three process behavior groups in Table 2 are all highly represented, any of them would be a suitable target for mimicry. For this evaluation, we decided to use the {DL,RD,WT,RN} functional group. While this functional group is also representative of most benign processes, the average number and ratio of operations is completely different when compared to ransomware. This functional group seems to be the worst-case scenario for our mimicry evasion technique. As illustrated in Table 4, for benign processes in the {DL,RD,WT,RN} group, the ratio between operations is 1:16:13:1. This means that for each DL operation, there are 16 RD, 13 WT, and 1 RN operations, respectively. Moreover, processes in this functional group access on average about 0.83% of the total number of user files in the system. We split our ransomware traces in the test set by following these averages and ratios, resulting in 170 mimicry ransomware processes, and successively query the classifier with each of them. We replicate this experiment for each of the 42 ransomware sample in our test set. None of the mimicry processes for any of the 42 ransomware is detected by the ShieldFS classifier.

6.2.1.4 Discussion It is worth noting the huge improvement gained with mimicry with respect to process splitting. In both mimicry and process splitting, each process performs all ransomware operations and therefore exhibits all the features used by ShieldFS for classification. However, with 170 process-split ransomware the detection rate of ShieldFS is about 14%, while with mimicry we are able to fully evade the detector. In comparison, process splitting needs almost two orders of magnitude more processes to achieve full evasion: 11,000 processes.

6.2.2 RWGuard

This section evaluates the effectiveness of our techniques against the RWGuard ransomware detector.

6.2.2.1 Process splitting We implement process splitting as in the evaluation against ShieldFS. As illustrated in Fig. 3b, the detection accuracy for RWGuard follows a curve similar to that of ShieldFS: the accuracy of the classifier initially remains stable around the original 99.4%, until a critical point, after which it quickly decreases to $\sim 10\%$. Afterward, both curves exhibit a long tail, with the detection accuracy very slowly decreasing to zero after 400 processes for RWGuard.

6.2.2.2 Functional splitting The RWGuard detector uses eight features to classify benign and malicious processes: RD, WT, OP, CL, FRD, FWT, FOP, and FCL. In this evaluation, we split the ransomware traces based on all eight features, and assess how each functional split

Table 4 Ratio between different operations for various types of benign processes

Combination	DL	RD	WT	RN	RD entropy	WT entropy	File access (%)
RD, RN	0	2	0	1	0.53	0	0.02
WT	0	0	1	0	0	0.42	0.60
DL, RD, WT, RN	1	16	13	1	0.59	0.46	0.83
RD	0	1	0	0	0.46	0	0.03
WT, RN	0	0	5	1	0	0.47	0.02
RD, WT	0	5	1	0	0.29	0.57	1.33
DL, RD, RN	8	39	0	1	0.42	0	0.09
DL, WT	2	0	1	0	0	0.51	0.01
RD, WT, RN	0	6	20	1	0.53	0.28	0.22
DL, RD, WT	3	52	1	0	0.57	0.77	0.17
DL	1	0	0	0	0	0	0.00
DL, RD	1	2	0	0	0.52	0	0.17
DL, WT, RN	1	0	8	2	0	0.39	0.03
DL, RN	45	0	0	1	0	0	0.06
RN	0	0	0	1	0	0	0.03

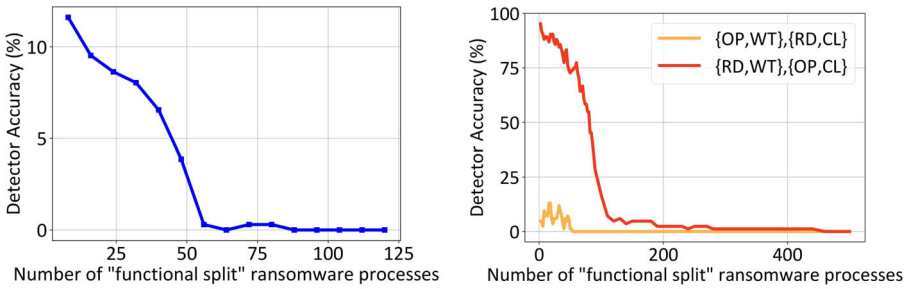


Fig. 5 Evaluation of the functional splitting evasion technique against RWGuard

(a) Single Functional Splitting. **(b)** Combined Functional Splitting.

ransomware process performs against the detector. We begin the evaluation with single functional splitting, where each functional split process performs only one type of operation, resulting in eight functional groups (one for each feature). Within each functional group, we apply process splitting until complete evasion is achieved. As shown in Fig. 5a, to fully evade the RWGuard classifier we need 64 functional split processes – 8 for each functional group.

We further study how different functional groupings affect the accuracy of RWGuard. In particular, we evaluate the accuracy of RWGuard against two different implementations of functional split. In the first, the operations are divided into the two functional groups {OP,WT},{RD,CL}. For the second implementation, we use the {RD,WT},{OP,CL} functional groups. For the purpose of grouping, we make no distinction between normal and fast operations in this experiment. As shown in Fig. 5b and consistently with our ShieldFS evaluation (Fig. 4b), we see that the initial accuracy for {RD,WT},{OP,CL} is much higher than in the single functional splitting case, starting at approximately 95% for two processes (one per functional group). This behavior is to be expected since RD and WT, two of the features with the highest importance for both detectors, are performed in the same functional group. Indeed, when we split these operations in two separate functional groups the accuracy of RWGuard is much lower, starting at $\sim$4% with only 2 processes ({OP,WT}, {RD,CL} in Fig. 5b).

6.2.2.3 Mimicry We evaluate our mimicry approach against the RWGuard classifier. As for the ShieldFS evaluation, we model ransomware features so they are, on average, identical to those of benign processes. In particular, we model the main features used by RWGuard: RD, WT, OP, CL, FRD, FWT, FOP, and FCL. We split the ransomware traces in the test set by following the average operation number and operation ratio performed by benign processes, which resulted in 10 mimicry ransomware processes, and queried the classifier with each individual split trace. None of the 42 ransomware samples in our test set was detected by RWGuard.

6.3 Cerberus evaluation

This section evaluates Cerberus, our ransomware prototype implementing functional splitting and mimicry evasion. Section 6.2 showed that our evasion techniques are effective when applied on the traces generated by traditional ransomware. However, it is still necessary to demonstrate that a prototype implementation of our techniques would work in practice. Moreover, in Sect. 6.2 we modeled mimicry processes based on the benign processes of the dataset used for the training of ShieldFS and RWGuard. It is necessary to determine whether our techniques can generalize to the case where the benign process model is derived from a surrogate dataset (i.e., a dataset different from the one used to train the classifier).

6.3.1 ShieldFS

We evaluate Cerberus against ShieldFS in our virtual machine, both in the functional split and mimicry modes. Cerberus implements functional splitting with the following three functional groups: {DL},{WT},{RD,RN}. Reading and renaming operations are performed by the same process group mainly for implementation convenience. It is worth noting that aggregating these two features makes it easier for the classifier to detect the ransomware. By setting Cerberus to use 6 processes per functional group (18 processes total, which is the closest to the 20 processes suggested by our trace-based evaluation), we were able to fully evade the detector: no functional split process was flagged as ransomware.

We also evaluate Cerberus in mimicry mode against ShieldFS. We described the details of the mimicry implementation in Cerberus in Sect. 5.1. The number of processes in mimicry mode depends on the average number of files accessed by the mimicked benign process group in our dataset. Table 4 shows that {DL,RD} processes access on average $\sim$0.17% of the total files, while {RD,WT,RN}

processes access $\sim 0.22\%$. In our VM, this results in a Cerberus run with 470 mimicry processes, which were all able to evade the ShieldFS detector, fully encrypting the VM files. This evaluation proves that our attacks are practical and applicable in realistic settings.

6.3.2 RWGuard

We further evaluate Cerberus against RWGuard in our virtual machine, both in the functional split and mimicry modes. An important difference compared to the RWGuard evaluation in Sect. 6.2 is that functional splitting in Cerberus considers only three functional groups, that are: {DL}, {WT}, and {RD,RN}. Cerberus does not split RWGuard-specific features (i.e., OP, CL, FOP, FCL, FRD, FWT). Regardless of this fact, we are able to fully evade RWGuard with Cerberus set to use 18 functional split processes in total (6 per functional group), as in the ShieldFS case.

We also evaluate the mimicry mode of Cerberus against RWGuard. For this evaluation, Cerberus is trained with the model of benign processes obtained from the ShieldFS dataset, while the RWGuard model is trained on the original dataset used by the authors in [5]. As before, in our VM evaluation Cerberus runs with 470 mimicry ransomware processes, which are all able to fully evade the RWGuard detector, fully encrypting the VM files. This evaluation shows that our evasion techniques can generalize to classifiers trained on different datasets.

6.4 Evaluation against a malwarebytes anti-ransomware

In previous experiments, the features used by the detectors (ShieldFS and RWGuard) were known. However, in a real attack scenario this white-box setting assumption might not hold true. The last part of our experimental evaluation focuses on black-box settings where details of the detectors are not known. In particular, we pitch Cerberus against a leading commercial ransomware detector: Malwarebytes Anti-Ransomware. Malwarebytes states that their Anti-Ransomware tool "does not rely on signatures or heuristics" [23], but rather leverages machine learning techniques [24]. We have no knowledge of the internal workings of the Malwarebytes classifier, such as which features it uses for classification, nor of its dataset. This makes Malwarebytes an ideal detector to test the viability of our evasion techniques in a black-box setting. We evaluate Cerberus against Malwarebytes in both the functional split and mimicry modes. For the functional splitting approach, we continue to set Cerberus to use a total of 18 functional split processes (6 per functional group). All 18 functional split processes successfully evade Malwarebytes, fully encrypting the VM files.

We also evaluate Cerberus running in mimicry mode against Malwarebytes Anti-Ransomware. As usual, the mimicry behavior of Cerberus processes is modeled based on the ShieldFS benign process dataset. Therefore, Cerberus runs with the usual 470 mimicry ransomware processes, which all successfully evade Malwarebytes and fully encrypt the VM files. This last experiment shows that our evasion techniques are general, are effective on commercial detectors, and work in a black-box setting where we have no information on the classifier.

6.5 Detecting functional splitting

In this section, we consider the problem of hardening a behavioral ransomware classifier against functional splitting. We focus on functional splitting as this attack is extremely effective, but results in behavior which, while avoiding detection by a traditional classifier, arguably differs significantly from that of benign processes. Therefore, differently from mimicry—which results in behavior which is, feature-wise, indistinguishable from benign processes—it is conceivable than an appropriately trained behavioral classifier may gain the ability to detect a functional splitting attack.

To evaluate the above hypothesis, we implemented a multi-tier random forest classifier using the ShieldFS architecture (see Sect. 2). For this evaluation, we performed two sets of experiments with different datasets:

1. Complete training set. The testing and training sets contain normal ransomware samples as well as functional split ransomware samples with up to 50 splits (i.e., 200 processes for single and 100 for combined functional splitting). With this experiment we evaluate whether the model is able to properly classify traditional ransomware as well as functional split ransomware with a number of splits seen during training (i.e., the testing and training set both contain samples split up to 50 times).
2. Partial training set. The testing set contains normal ransomware samples as well as functional split ransomware samples with up to 50 splits. The training set contains normal ransomware samples as well as functional split ransomware samples with up to 10 splits. With this experiment we evaluate whether the model is able to generalize, properly classifying functional split ransomware with a number of splits that are much higher than those seen during training.

Figure 6a, b illustrate our results when training with the complete dataset (1), for single functional splitting and

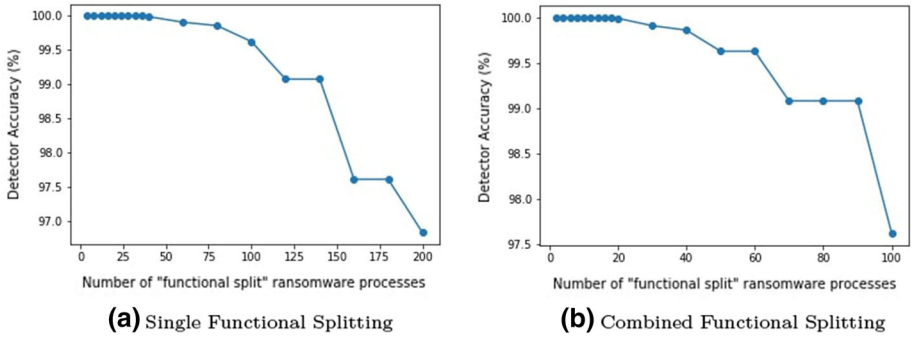


Fig. 6 Evaluation of the functional splitting evasion technique against a random forest detector trained on functional split ransomware with up to 200 splits

combined functional splitting, respectively. As we can see, the performance of the detector remains consistently above $\sim 97\%$ for both single and combined splitting for up to 50 splits (i.e., 200 processes for single and 100 for combined functional splitting). This is in contrast with the original results of the ShieldFS classifier, where detection accuracy quickly reached 0 after only 5 splits for single functional splitting and 10 for combined. These results indicate that the heavy drop in performance found in the original model (see Sect. 6.2) is due to the lack of representation of functional split ransomware in the dataset, rather than because of intrinsic limitations of behavioral models in detecting functional split ransomware.

Overall, Fig. 6 highlights that a model trained on a *specific* number of splits is able to detect ransomware that is split that specific number of times. However, it does not directly address the question of whether such behavioral model is robust and can generalize in terms of the number of splits. In particular, in a real-world setting it is not possible to know in advance how many times a ransomware will be split, and train the model accordingly. A robust ransomware detector should be able to generalize and detect ransomware beyond what was seen during training. Figure 7 shows the performance of the behavioral model trained with the partial training set (2), for both single and combined functional splitting. As discussed earlier, the partial training set contains only functional split ransomware that is split up to 10 times, while the testing set contains ransomware split up to 50 times. As we can see from the figure, the model is able to maintain good detection performance for ransomware with a number of split that is well beyond what is seen during training. In particular, the model maintains a 100% detection accuracy for ransomware split up to 10 times, with an expected drop in performance for higher (unseen) splits, decreasing to 92% for 25 splits and to $\sim 86\%$ for 50 splits. This behavior strongly indicates that an appropriately trained behavioral model can detect with near-perfect accuracy known functional split ransomware, as well as generalize on unseen ransomware with a number of splits not seen during training.

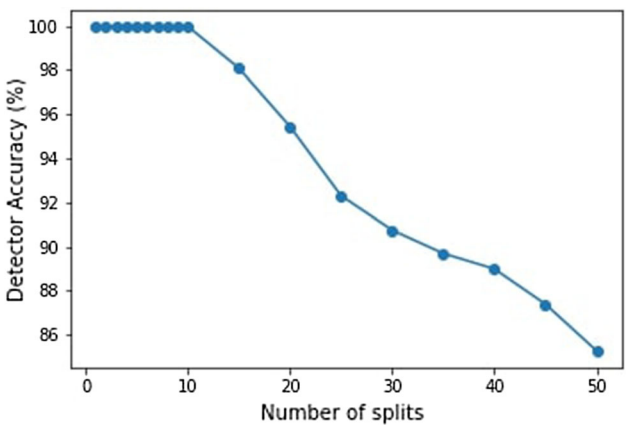


Fig. 7 Evaluation of the functional splitting evasion technique against a random forest detector trained on functional split ransomware with up to 10 splits

The results presented in this section validate our hypothesis in Sect. 3.3, that behavioral classifiers can be trained to recognize the distinctive trace of functional splitting and, therefore, that a more complex attack like mimicry is required.

6.6 Limitations

While our proposed attacks can be extremely powerful when designed correctly, there are some limitations that can apply in real-world applications. The main limitation of our family of attacks is that they target a predefined set of features for evasion. While our black-box evaluation against Malwarebytes Anti-Ransomware shows that our techniques are effective even without knowledge of the set of features used by the detector, there needs to be at least a partial overlap in the features that are targeted for evasion and the features used by the detector. In real-world settings, when no knowledge about the target detector is available, the number of potential features that must be considered can become potentially large. For functional splitting, a large number of features to evade means more functional groups required for the splitting, which results in a larger number of processes. For mimicry, an increasing number of

target features to evade requires increasingly fine-grained modeling of benign processes, which can become challenging. Moreover, if the dataset of benign processes is not big enough, the modeled behaviors might not be general enough and can differ from the behavior of benign processes on the target system, potentially decreasing the effectiveness of the attack. Finally, modeling error and uncertainties related to real systems can further influence the effectiveness of mimicry [32, 33].

7 Countermeasures

Graph-based malware detection approaches work by building a *provenance graph*, which represent data and control flow relationships between processes and operating system entities (files, sockets, memory) on a given machine. Such graphs are then analyzed to detect suspicious behavior using either rules [34] or unsupervised anomaly detection [35, 36]. These techniques have been successfully applied to the detection of APTs across long timescales and different machines. While in principle we believe that information-flow correlation between processes is an interesting direction for a countermeasure, current proposals have limitations. While these techniques are successful in detecting APTs, they typically do so only after multiple (or all) stages of the APT have completed. While this is acceptable for APTs since the goal is to eventually reveal their presence, ransomware requires immediate detection and swift remediation *before* the ransomware encrypts user data. Moreover, unsupervised approaches tend to have low accuracy on machines with unpredictable, varied workloads—such as user workstations [36], which are often ransomware targets. Therefore, we believe further work is necessary to adapt graph-based threat detection to the class of attacks described here.

Another approach entails identifying *synchronized process behavior* across applications running concurrently in different machines. This approach leverages the insight that a ransomware infection typically involves an entire network. Similar approaches, although based on network traffic, have proven effective for botnet nodes detection [37]. We note that both the functional splitting and mimicry attack can, by design, split operations in arbitrarily different ways. This enables randomizing the attack behavior across different machines.

Several works analyze the applicability of Hardware Performance Counters (HPC) to resilient malware detection [38, 39]. HPC are hardware components that record behaviors at the micro-architectural level, such as for instance load/store operations, cache hits, and misses or correct/incorrect branch predictions. The rationale is that malware exhibits very distinct HPC profiles compared to benign software, and therefore machine learning can be applied to HPC data to reliably classify benign and malicious applications, including ransomware. While early works indicated promising results, recent research has shown that HPC-based approaches are much less effective than previously suggested and cannot be reliably used to distinguish malicious software [40, 41].

Finally, many defenses against adversarial attacks—both theoretical and practical—have been proposed (e.g., [42, 43]). We posit that the current generation of behavioral malware detectors exhibits *feature vulnerability* [21], i.e., it is possible to generate malicious behavior that looks, feature-wise, exactly like benign behavior. This suggests that increasing the sophistication of the classifier without rethinking the features, may not suffice to remediate such attacks.

8 Related work

8.1 Ransomware detection

For a review of behavioral ransomware detection techniques [4–7, 22, 23], the reader is referred to Sect. 2.2. Other proposals focus specifically on randomness of written data to identify encrypted content. Data-aware Defense [44] performs the χ^2-test on a sliding window of write operations. Mbol et al. [45] use a test based on the Kullback-Liebler divergence to detect ransomware converting high-entropy JPEG files to encrypted content. Depending exclusively on randomness is dangerous [46, 47] for reasons pointed out in Sect. 4.1. An orthogonal line of work focuses on *decoy files* for ransomware detection [48–50]. Such defenses are outside the scope of our work. We believe decoys are a promising strategy, but they raise usability concerns, and their evasion has been poorly studied. Finally, for a discussion of relevant graph-based detection approaches [34–36] see Sect. 7.

8.2 Multiprocessing in existing malware

Several existing ransomware families use multi-processing. This happens for example in WannaCry and Petya. Encryption is still performed by one process, while the others perform non encryption-related auxiliary tasks [51, 52]. The CERBER ransomware (not to be confused with our *Cerberus* prototype), despite its name, does not appear to perform multi-process encryption. While it has been claimed that CERBER attempts to evade machine learning, these claims refer to obfuscation of static payload features [53]. MalWASH [25] and its successor D-TIME [30] split the malware code into chunks and inject

an emulator to execute them across a set of benign processes. This approach would generate a significant overhead for compute-intensive ransomware activity. Conversely, we found that multi-process splitting, combined with mimicry, generates near-zero overhead and suffices to avoid detection.

8.3 Evasion of ransomware detectors

The work closest in spirit to ours is the critical analysis of ransomware defenses by Genç et al. [54]. For what concerns behavioral detection, their work is more limited in scope than ours and consider a smaller set of features. Furthermore, the work by Genç et al does not incorporate the notion of mimicry and only focus on simple feature obfuscation (e.g., avoid changing file types).

8.4 Adversarial sample generation

Generation of adversarial samples for various classes of malicious programs has been studied. In the mobile malware domain, Grosse et al. [20] generate malicious Android app packages which go undetected by a custom neural network classifier which uses manifest-derived features. This attack uses Papernot's method [13] to guide mutations to app manifests.

Other attacks are not specifically focused on mobile. Anderson et al. [55] propose the use of reinforcement learning to guide mutations to malicious executables to make them undetectable. Rosenberg et al. [56] generate adversarial malware binaries by altering various static features using a custom mimicry attack. Hu and Tan [57] propose the use of generative adversarial networks (GANs, which have been used in a wide range of applications [58, 59]) to mutate feature vectors derived from the presence/absence of imported DLLs and API calls in a malware binary. [57] does not propose a method to concretely generate malware binaries, only feature vectors. Finally, several works propose attacks which mutate the static structure of PDF-based exploits to prevent their detection [15–17, 60]. All the works above focus on static features, i.e., they alter the appearance of a malicious file object, but not its run-time behavior.

There is limited work on attacking dynamic (behavioral) features—i.e., features generated by actions performed by a process at run-time. Arguably, such features are harder to attack; indeed, they represent actions that a malware needs to execute in order to achieve its malicious goals. Rosenberg et al. [56, 61] and Hu and Tan [62] proposed methods to defeat malware detectors trained on dynamically-generated sequences of API calls. These proposals work by perturbing the sequence of API calls, chiefly by inserting dummy calls. While we use dummy calls as part of our mimicry attack, we also leverage a broader set of capabilities such as distributing calls across processes to reduce feature expression. This give our technique the ability to decrease per-process frequencies/counts of certain calls without slowing down the attack (necessary to defeat the detectors in our evaluation), or to obfuscate data dependencies between calls (such dependencies are used by some detectors, e.g., [63]).

9 Ethical considerations

Our work is motivated by the interest in understanding the limitations of current malware detectors. We believe that doing so is necessary to ensure that detection algorithms remain effective against the constantly-shifting threat landscape. We **do not** plan to publicly release our ransomware prototype (Sect. 5.1), in order to prevent its use in threat development. However, we do plan to make it available on a case-by-case basis to reputable research groups. Prior to publishing our previous conference paper [11], we also communicated details and results of our attack to Malwarebytes.

10 Conclusions

We proposed and analyzed a novel practical attack against behavioral malware detectors. Our attack splits malware operations across a set of cooperating processes, in such a way that no individual process behavior is flagged as suspicious by a behavioral process classifier.

We concretely defined and implemented this concept in the ransomware domain. We proposed three novel attacks, *process splitting*, *functional splitting*, and *mimicry*. Our methods successfully evade state-of-the-art detectors without limiting the capabilities of ransomware. However, we also find that adversarial training is effective against functional splitting.

To the best of our knowledge, this is the first comprehensive evaluation of this attack model, and possible defenses, in the domain of malware and ransomware in particular. Our work highlights significant limitations in behavioral detection algorithms and has relevant practical implications for the current generation of malware detectors.

Funding Open access funding provided by Alma Mater Studiorum - Universitá di Bologna within the CRUI-CARE Agreement. This work was supported by Gen4olive, a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101000427, and in part by the Italian MIUR through the Dipartimento di Informatica, Sapienza

University of Rome, under Grant Dipartimenti di eccellenza 2018–2022.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Moser A, Kruegel C, Kirda E (2007) Limits of static analysis for malware detection. In: Twenty-Third annual computer security applications conference (ACSAC 2007), IEEE, pp 421–430
2. O'Kane P, Sezer S, McLaughlin K (2011) Obfuscation: the hidden malware. IEEE Secur Priv 9(5):41–47
3. Tian R, Islam R, Batten L, Versteeg S (2010) Differentiating malware from cleanware using behavioural analysis. In: 2010 5th international conference on malicious and unwanted software, IEEE, pp 23–30.
4. Continella A, Guagnelli A, Zingaro G, De Pasquale G, Barenghi A, Zanero S, Maggi F (2016) Shieldfs: a self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd annual conference on computer security applications, pp 336–347
5. Mehnaz S, Mudgerikar A, Bertino E (2018) Rwguard: a real-time detection system against cryptographic ransomware. In: International symposium on research in attacks, intrusions, and defenses, Springer, pp 114–136.
6. Kharaz A, Arshad S, Mulliner C, Robertson W, Kirda E (2016) {UNVEIL}: A {Large-Scale}, automated approach to detecting ransomware. In: 25th USENIX security symposium (USENIX Security 16), pp 757–772
7. Kharraz A, Kirda E (2017) Redemption: real-time protection against ransomware at end-hosts. In: International symposium on research in attacks, intrusions, and defenses, Springer, pp. 98–119
8. Piskozub M, De Gaspari F, Barr-Smith F, Mancini L, Martinovic I (2021) Malphase: fine-grained malware detection using network flow data. In: Proceedings of the 2021 ACM Asia conference on computer and communications security, pp 774–786
9. Atlanta Spent \$2.6M to Recover From a \$52,000 Ransomware Scare. https://www.wired.com/story/atlanta-spent-26m-recover-from-ransomware-scare/ (2018)
10. WannaCry cyber attack cost the NHS £92m as 19,000 appointments cancelled. https://www.telegraph.co.uk/technology/2018/10/11/wannacry-cyber-attack-cost-nhs-92m-19000-appointments-cancelled/ (2018)
11. Gaspari FD, Hitaj D, Pagnotta G, Carli LD, Mancini LV (2020) The naked sun: malicious cooperation between benign-looking processes. In: International conference on applied cryptography and network security, Springer, pp 254–274
12. Biggio B, Rieck K, Ariu D, Wressnegger C, Corona I, Giacinto G, Roli F (2014) Poisoning behavioral malware clustering. In: Proceedings of the 2014 workshop on artificial intelligent and security workshop, pp 27–36
13. Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A (2016) The limitations of deep learning in adversarial settings. In: 2016 IEEE European symposium on security and privacy (EuroS&P), IEEE, pp 372–387
14. Kantchelian A, Tygar JD, Joseph A (2016) Evasion and hardening of tree ensemble classifiers. In: International conference on machine learning, PMLR, pp 2387–2396
15. Xu W, Qi Y, Evans D (2016) Automatically evading classifiers: a case study on pdf malware classifiers. In: NDSS
16. Laskov P *et al.* (2014) Practical evasion of a learning-based classifier: a case study. In: 2014 IEEE symposium on security and privacy, IEEE, pp 197–211
17. Biggio B, Corona I, Maiorca D, Nelson B, Šrndić N, Laskov P, Giacinto G, Roli F (2013) Evasion attacks against machine learning at test time. In: Joint European conference on machine learning and knowledge discovery in databases, pp 387–402
18. Yang W, Kong D, Xie T, Gunter CA (2017) Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In: Proceedings of the 33rd annual computer security applications conference, pp 288–302
19. Demontis A, Melis M, Biggio B, Maiorca D, Arp D, Rieck K, Corona I, Giacinto G, Roli F (2017) Yes, machine learning can be more secure! a case study on android malware detection. IEEE Trans Dependable Secur Comput 16(4):711–724
20. Grosse K, Papernot N, Manoharan P, Backes M, McDaniel P (2017) Adversarial examples for malware detection. In: European symposium on research in computer security, Springer, pp 62–79
21. Maiorca D, Demontis A, Biggio B, Roli F, Giacinto G (2020) Adversarial detection of flash malware: limitations and open issues. Comput Secur 96:101901
22. Scaife N, Carter H, Traynor P, Butler KR (2016) Cryptolock (and drop it): stopping ransomware attacks on user data. In: 2016 IEEE 36th international conference on distributed computing systems (ICDCS), IEEE, pp 303–312
23. Introducing the Malwarebytes Anti-Ransomware Beta. https://blog.malwarebytes.com/malwarebytes-news/2016/01/introducing-the-malwarebytes-anti-ransomware-beta/ (2016)
24. Malwarebytes Anti-Ransomware for Business. https://www.malwarebytes.com/business/solutions/ransomware/ (2019)
25. Ispoglou KK, Payer M (2016) {malWASH}: washing malware to evade dynamic analysis. In: 10th USENIX workshop on offensive technologies (WOOT 16)
26. Lin J (1991) Divergence measures based on the Shannon entropy. IEEE Trans Inf Theor 37(1):145–151
27. List of File Signatures. https://en.wikipedia.org/wiki/List_of_file_signatures (2019)
28. Bellare M, Ristenpart T, Rogaway P, Stegers T (2009) Format-preserving encryption. In: International workshop on selected areas in cryptography, Springer, pp 295–312
29. I/O request packets. https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/i-o-request-packets (2017)
30. Pavithran J, Patnaik M, Rebeiro C (2019) {D-TIME}: distributed threadless independent malware execution for runtime obfuscation. In: 13th USENIX workshop on offensive technologies (WOOT 19)
31. Lv Z, Zhao Y, Zhang C, Li H (2020) Dramd: detect advanced dram-based stealthy communication channels with neural networks. In: IEEE INFOCOM 2020-IEEE Conference on computer communications, IEEE, pp 1907–1916

32. Chen Z, Zhang B, Stojanovic V, Zhang Y, Zhang Z (2020) Event-based fuzzy control for TS fuzzy networked systems with various data missing. Neurocomputing 417:322–332
33. Cheng P, He S, Stojanovic V, Luan X, Liu F (2021) Fuzzy fault detection for markov jump systems with partly accessible hidden information: an event-triggered approach. IEEE transactions on cybernetics
34. Milajerdi SM, Gjomemo R, Eshete B, Sekar R, Venkatakrishnan V (2019) Holmes: real-time apt detection through correlation of suspicious information flows. In: 2019 IEEE symposium on security and privacy (SP), IEEE, pp 1137–1152
35. Manzoor E, Milajerdi SM, Akoglu L (2016) Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1035–1044
36. Han X, Pasquier T, Bates A, Mickens J, Seltzer M (2020) Unicorn: runtime provenance-based detector for advanced persistent threats. In: Proceedings of the 2020 network and distributed system security symposium
37. Gu G, Porras PA, Yegneswaran V, Fong MW, Lee W (2007) Bothunter: detecting malware infection through ids-driven dialog correlation. In: USENIX security symposium, vol. 7, pp 1–16
38. Demme J, Maycock M, Schmitz J, Tang A, Waksman A, Sethumadhavan S, Stolfo S (2013) On the feasibility of online malware detection with performance counters. ACM SIGARCH Comput Archit News 41(3):559–570
39. Khasawneh KN, Abu-Ghazaleh N, Ponomarev D, Yu L (2017) Rhmd: evasion-resilient hardware malware detectors. In: Proceedings of the 50th annual IEEE/ACM international symposium on microarchitecture, pp 315–327
40. Zhou B, Gupta A, Jahanshahi R, Egele M, Joshi A (2018) Hardware performance counters can detect malware: myth or fact? In: Proceedings of the 2018 on Asia conference on computer and communications security, pp 457–468
41. Das S, Werner J, Antonakakis M, Polychronakis M, Monrose F (2019) Sok: the challenges, pitfalls, and perils of using hardware performance counters for security. In: 2019 IEEE symposium on security and privacy (SP), IEEE, pp 20–38
42. Tong L, Li B, Hajaj C, Xiao C, Vorobeychik Y (2017) Hardening classifiers against evasion: the good, the bad, and the ugly. CoRR, arXiv:1708.08327
43. Carlini N, Wagner D (2017) Adversarial examples are not easily detected: bypassing ten detection methods, pp 3–14
44. Palisse A, Durand A, Le Bouder H, Le Guernic C, Lanet J-L (2017) Data Aware Defense (DaD): Towards a Generic and Practical Ransomware Countermeasure. In: Secure IT systems vol. 10674, Springer, Cham, pp 192–208
45. Mbol F, Robert J-M, Sadighian A (2016) An Efficient Approach to Detect TorrentLocker Ransomware in Computer Systems. In: Cryptology and Network Security vol. 10052, pp. 532–541. Springer, Cham
46. De Gaspari F, Hitaj D, Pagnotta G, De Carli L, Mancini LV (2020) Encod: distinguishing compressed and encrypted file fragments. In: Network and system security, pp 42–62
47. De Gaspari F, Hitaj D, Pagnotta G, De Carli L, Mancini LV (2021) Reliable detection of compressed and encrypted data. arXiv preprint arXiv:2103.17059
48. Genç ZA, Lenzini G, Sgandurra D (2019) On deception-based protection against cryptographic ransomware. In: International conference on detection of intrusions and malware, and vulnerability assessment, Springer, pp 219–239
49. Moore C (2016) Detecting ransomware with honeypot techniques. In: CCC
50. Moussaileb R, Bouget B, Palisse A, Le Bouder H, Cuppens N, Lanet J-L (2018) Ransomware's early mitigation mechanisms. In: Proceedings of the 13th international conference on availability, reliability and security, pp 1–10
51. WannaCry Analysis and Cracking. https://medium.com/@codingkarma/wannacry-analysis-and-cracking-6175b8cd47d4 (2018)
52. "Petya-like" Ransomware Analysis. https://www.nyotron.com/wp-content/uploads/2017/06/NARC-Report-Petya-like-062017-for-Web.pdf (2017)
53. Cerber Starts Evading Machine Learning. https://blog.trendmicro.com/trendlabs-security-intelligence/cerber-starts-evading-machine-learning/ (2017)
54. Genç ZA, Lenzini G, Ryan PYA (2018) Next generation cryptographic ransomware. In: Secure IT systems vol. 11252, Springer, Cham, pp 385–401.
55. Anderson HS, Kharkar A, Filar B, Roth P (2017) Evading machine learning malware detection 2017:6
56. Rosenberg I, Shabtai A, Rokach L, Elovici Y (2018) Generic black-box end-to-end attack against state of the art API call based malware classifiers. In: International symposium on research in attacks, intrusions, and defenses, Springer, pp 490–510
57. Hu W, Tan Y (2017) Generating adversarial malware examples for black-box attacks based on GAN. arXiv:1702.05983 [cs] . arXiv: 1702.05983. Accessed 2018-09-07
58. Hitaj B, Gasti P, Ateniese G, Perez-Cruz F (2019) Passgan: a deep learning approach for password guessing. In: International conference on applied cryptography and network security, pp 217–237
59. Pagnotta G, Hitaj D, De Gaspari F, Mancini LV (2021) Passflow: guessing passwords with generative flows. arXiv preprint arXiv:2105.06165
60. Dang H, Huang Y, Chang E-C (2017) Evading classifiers by morphing in the dark. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp 119–133
61. Rosenberg I, Shabtai A, Elovici Y, Rokach L (2018) Query-efficient gan based black-box attack against sequence based machine and deep learning classifiers. arXiv:1804.08778 [cs]. Accessed 2018-11-01
62. Hu W, Tan Y (2018) Black-box attacks against RNN based malware detection algorithms
63. Fredrikson M, Jha S, Christodorescu M, Sailer R, Yan X (2010) Synthesizing near-optimal malware specifications from suspicious behaviors. In: 2010 IEEE symposium on security and privacy, IEEE, pp 45–60