The Naked Sun: Malicious Cooperation Between Benign-Looking Processes

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

[1] Dipartimento di Informatica, Sapienza Università di Roma, Italy
{degaspari, hitaj.d, pagnotta, mancini}@di.uniroma1.it,
[2] Department of Computer Science, Worcester Polytechnic Institute
ldecarli@wpi.edu

Abstract. Recent progress in machine learning has generated promising results in behavioral malware detection, which identifies malicious processes via features derived by their runtime behavior. Such features hold great promise as they are intrinsically related to the functioning of each malware, and are therefore 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 thoroughly examines the robustness of behavioral ransomware detectors to evasion. Ransomware behavior tends to differ significantly from that of benign processes, making it a low-hanging fruit for behavioral detection (and a difficult candidate for evasion). Our analysis identifies a set of novel attacks that distribute the overall malware workload across a small set of cooperating processes to avoid the generation of significant behavioral features. Our most effective attack decreases the accuracy of a state-of-the-art detector from 98.6% to 0% using only 18 cooperating processes. Furthermore, we show our attacks to be effective against commercial ransomware detectors.

Keywords: Malware · Ransomware · Evasion of threat detection.

1 Introduction

Malware detection is a difficult problem, with no full solution in sight despite decades of research. The traditional approach—based on analysis of static signatures of the malware binary—is increasingly rendered ineffective by polymorphism and program obfuscation tools [32,34]. Using such tools, malware creators can quickly generate thousands of binary variants of functionally identical samples, effectively circumventing signature-based approaches.

We thank the authors of RWGuard for sharing their code with us and the authors of ShieldFS for sharing their dataset and helping us re-implement their detector. Fabio De Gaspari, Dorjan Hitaj, Giulio Pagnotta and Luigi V. Mancini are supported in part by the Italian MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Dipartimento di Informatica of Sapienza University of Rome.

As a result, the focus of the community has increasingly shifted towards 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. At first sight, these techniques seem to hold great promise: the behavior of malware differs significantly from that of benign processes and this marked difference can be exploited to differentiate between these two classes of processes. In particular, recent improvements in the field of Machine Learning (ML) showed that ML models are extremely effective in distinguishing between different behavioral classes, resulting in very high accuracy [12, 29]. 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 [12,24,25,29].

Despite the success of ML-based behavioral analysis, a growing body of work has cast a shadow over the robustness of ML in adversarial settings [11, 17]. In this work, we assesses the robustness of recently-proposed behavioral-based ransomware detection tools [12, 29]. We use ransomware as a case study due to both the gravity of the threat (e.g., [5, 7]), 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 the tool's inner workings, its 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[3]. 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. **Our 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 ransomware following our approach, proving that our evasion technique is practical.

[3] In Isaac Asimov's 1957 novel *The Naked Sun*, a crime is committed by robots which are forbidden, by their programming, to harm humans. Each robot performs an apparently innocuous action, however the combination results in murder.

- We evaluate the proposed evasion techniques against multiple state-of-the-art behavioral 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 discuss the applicability of potential countermeasures.

2 Behavioral Ransomware Detection

The literature presents several recent works on ransomware detection based on behavioral features [9, 12, 24, 29, 38]. UNVEIL [9] and its successor Redemption [24] detect suspicious activity by computing a score using a 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 [38] 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.

In this paper, we demonstrate (i) heuristics to generate ransomware behavior which goes undetected by behavioral detectors, and (ii) a proof-of-concept ransomware prototype implementing these heuristics. Our attack is motivated by a review of all the approaches cited above; for evaluation we selected two of them, 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).

ShieldFS [12] identifies ransomware processes at file-system level and transparently rolls 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, using a hierarchy of random forest classifiers tuned at different temporal resolutions. This approach allows ShieldFS to take into account both short- and long-term process history when performing classification. ShieldFS uses features typically associated with ransomware operation for the classifiers, 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 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

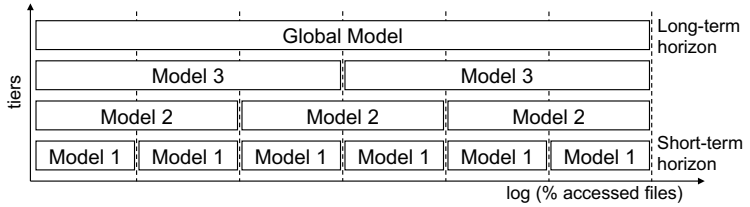


Fig. 1. Incremental models in ShieldFS (reproduced from [12])

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 [12]) shows the 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 feature values across all processes in the system. This is however only used to disambiguate ambiguous results from per-process classifiers. When our attack is successful, individual processes are always classified as benign with high confidence and therefore the system-wide classifier is never triggered.

RWGuard [29] 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. RWGuard uses a relatively simple detector consisting of a random forest classifier that analyzes process behavior using a 3 seconds 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 consists of three steps: when the behavioral classifier detects a suspicious process activity, the file monitor component is invoked to validate the detection. If both modules agree that the activity is suspicious, a File Classification module is invoked to further 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.

Malwarebytes Several commercial anti-ransomware solutions exist; for our work, we chose to evaluate Malwarebytes' Anti-Ransomware [1]. Differently from

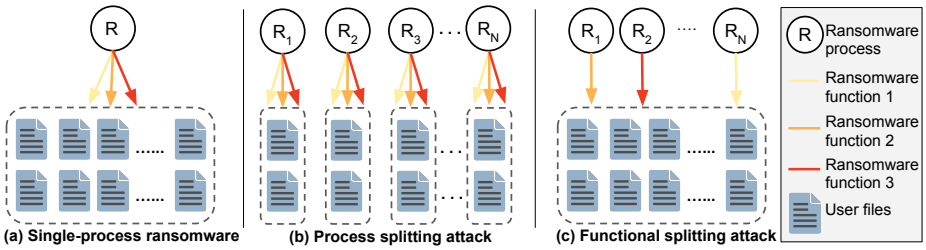


Fig. 2. Process and functional splitting attacks

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" [1] and leverages machine learning [8]. 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 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. 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 by devising an evasion technique based on distributing the malware operations across multiple *independent* processes: each process individually appears to have a benign behavior. However, the aggregated action of all these processes results in the intended malicious 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 unrelated processes working independently to achieve a common goal [23]. 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) [23]. We omit a full discussion of inter-process covert channels as it is outside the scope of this paper. The remainder of this section describes our three proposed attacks.

3.1 Process Splitting

In process splitting (depicted in Figure 2b), the ransomware behavior is distributed 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. 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 (Section 6) show it can be extremely effective. Given a target classifier, the number of ransomware processes can be arbitrarily increased 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, 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* (Figure 2c). Ransomware processes perform a set of operations (or *functions*) to encrypt user files, such as reading, writing or directory listing. In 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 each individual process. The intuition behind the functional splitting approach is that classifiers use groups of features to classify processes. If a process only exhibits a small subset of these features, then it will not be classified as malicious. Functional splitting ensures that each functional split ransomware process only exhibit 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 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.

To avoid this potential drawback, 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 are ransomware processes that act exactly like benign processes. However, the collective behavior of all the mimicry processes results in the desired malicious 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 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.

Due to space limitations, in this section we limit the discussion to what we found to be the most used and robust features employed by current ransomware behavioral classifiers. Other file access-based features exist, but they can be evaded using techniques similar to those detailed below.

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 is a pseudorandom string with no structure, and exhibits maximum entropy [29], while structured data written by benign programs is assumed to have considerably lower entropy. Consequently, entropy of write operations appears to be a useful feature to differentiate ransomware from benign processes. All state-of-the-art ML ransomware detectors use entropy of write operations as a feature [12, 24, 25, 29].

Evasion: Entropy as a feature for ransomware detection can be used at different levels of granularity: (1) overall file entropy [29], (2) average read-write operations difference [24, 25], and (3) individual write operations [12]. Feature (1) does not allow accurate differentiation between ransomware and benign processes, as nowadays most common file types are compressed for efficiency, including pdf, docx, xlsx, video and image files. Consequently, the overall estimate of Shannon

entropy [26] for these file types is comparable to 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 (see Section 6.1) 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.

4.2 File Overwrite

One feature that is common across ransomware families is that original user files are fully overwritten, either with the encrypted data or with random data to perform a secure delete [25]. 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 involves limiting the percentage of a file overwritten by a single ransomware process. Maintaining this percentage within the range exhibited by benign processes can be easily achieved with our proposed multi-process ransomware. It is sufficient to distribute write operations to a given file over multiple processes. Each individual process does not show any suspicious behavior, but the aggregated action of all the processes still overwrites the whole file.

4.3 Read/Write/Open/Create/Close Operations

Ransomware tends to access and encrypt as many files as possible on a victim machine to maximize the damage. 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: 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, it is possible to limit how many file operations each individual ransomware process performs.

4.4 File Similarity

Ransomware always completely changes the content of a file by encrypting it. Conversely, benign processes rarely perform whole-file alterations. Therefore, overall file similarity before and after write operations from a given process is a strong feature to detect ransomware operation [38].

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	

Table 1. Notation

DL	RD	WT	RN	% of Processes
✓	✓	✓	✓	19.07
✓	✓	-	-	18.37
-	✓	✓	✓	16.35
-	✓	-	-	11.44
✓	✓	✓	-	7.60
-	✓	-	✓	6.85
-	-	-	✓	6.21
-	✓	✓	-	5.61

Table 2. Most represented behavioral profiles exhibited by benign processes

Evasion: This feature can be evaded similarly to 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.

5 Implementation: The Cerberus Prototype

Here, we briefly describe Cerberus, a new ransomware prototype for Windows developed to demonstrate the feasibility of our evasion techniques. Cerberus implements both *functional splitting* and *mimicry*. 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. We could have considered additional features for the implementation of functional splitting. 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.

To implement the mimicry attack in Cerberus, we performed a statistical analysis on the behavior of benign processes from the ShieldFS dataset (ref. Section 6.1). 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 rep-

Type	Benign	Ransomware
#Unique Applications	2245	383
#Applications Training Set	2074	341
#Applications Testing Set	171	42
#IRPs [Millions]	1763	663.6

Table 3. Dataset details

resented in the dataset of benign processes, as well as because no ransomware process belongs to any of these two classes. 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.

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? (Section 6.2); **(2)** Can our theoretical attack evade detection when implemented in a real-world setting? (Section 6.3); **(3)** Do our evasion techniques generalize, evading classifiers trained on different datasets? (Section 6.3); **(4)** Is our attack effective in a black-box setting against commercial behavioral ransomware detectors? (Section 6.4).

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 [12]. 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 Cerberus, we created a realistic virtual machine 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. Our VM is comprised of 33625 files for a total of $\sim 10GB$, distributed over 150 folders.

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 [3] of real ransomware from the ShieldFS dataset [12]. 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.

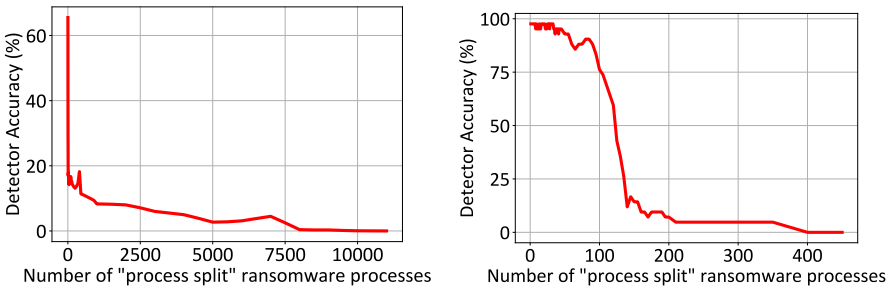
Our evaluation simulates multiple processes by splitting the IRP trace of a single ransomware in multiple traces, based on each evasion technique. Successively, we compute the feature vector for each individual trace as if it were an individual ransomware process. Finally, we query the classifier and 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.

ShieldFS - Process Splitting: As mentioned in Section 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. ShieldFS accuracy decreases already after a single split, going from single-process 98.6% accuracy down to 65.5% on a two-process ransomware[4]. Further splitting incurs diminishing returns. 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.

ShieldFS - Functional Splitting: 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.

First we 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 iteratively apply our process splitting technique until complete evasion is achieved.

[4] Our experiments only consider the ability of detectors to correctly classify ransomware processes. Since we do not consider benign processes, there can be no false positives nor true negatives. Therefore, for all our experiments accuracy is equivalent to true positive rate.

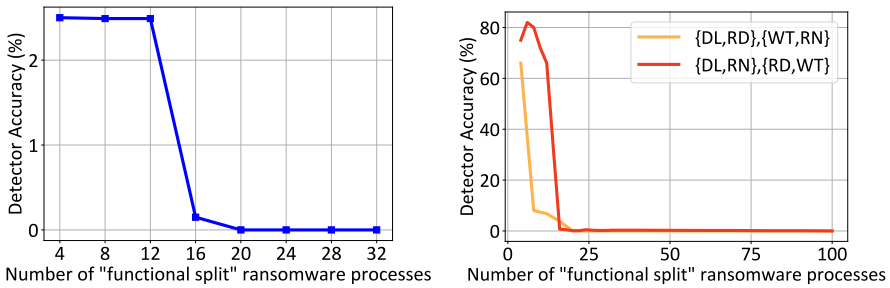


(a) ShieldFS (b) RWGuard

Fig. 3. Evaluation of the process splitting evasion technique

Figure 4a shows that 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 Figure 4a and Figure 3a. 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 ~ 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 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}. We chose these groups of operations due to their frequent combined appearance in our dataset (see Table 2). 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 Figure 4b is due to the fact that in the first ransomware implementation we have the {RD,WT} functional group and in the second imple-



(a) Single functional splitting (b) Combined Functional Splitting

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

mentation 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 towards zero.

ShieldFS - Mimicry: In the mimicry attack, we model ransomware features so that, on average, they are identical to those of benign processes. We build our model of a typical benign process by statistical analysis of the behavior of benign processes in the ShieldFS dataset [12], which comprises observations of well above one 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.

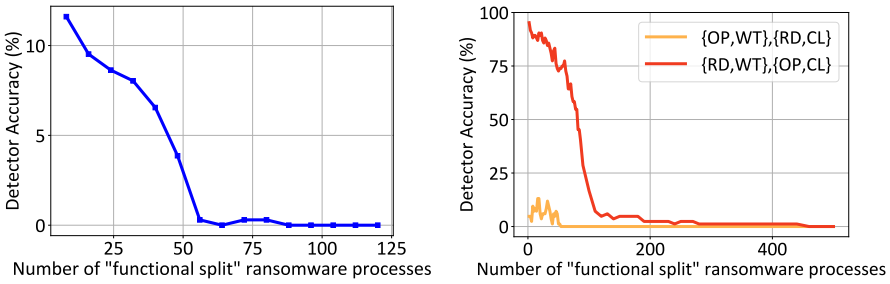
We focus on modeling the four main operations performed by ransomware and benign processes—DL, RD, WT, RN—together with the number of file accessed. Note that we could easily consider more features in our modeling, up to all features described in Section 4. However, since the goal of this evaluation 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 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 are detected by the ShieldFS classifier.
- 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, mimicry requires almost two orders of magnitude less processes to achieve full evasion (170 vs 11000).

RWGuard - Process Splitting: We implement process splitting as in the evaluation against ShieldFS. As illustrated in Figure 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 $\sim 10\%$. Afterwards, both curves exhibit a long tail, with RWGuard detection accuracy decreasing to zero after 400 processes.

RWGuard - 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 ransomware process performs against the detector. We begin the evaluation with each functional split process performing 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 Figure 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 accuracy. 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 Figure 5b and consistently with our ShieldFS evaluation (Figure 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. When we split these operations in two separate



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

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

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%

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

functional groups the accuracy of RWGuard is much lower, starting at $\sim$ 4% with only 2 processes ({OP,WT},{RD,CL} in Figure 5b).

RWGuard - Mimicry: As for the ShieldFS mimicry 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 were detected by RWGuard.

6.3 Cerberus Evaluation

This section evaluates our Cerberus prototype, demonstrating that our attacks are effective in practical settings. Furthermore, we show that 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).

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}. 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.

We also evaluate Cerberus in mimicry mode (ref. Section 5). 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. Thus our attacks are practical in realistic settings.

RWGuard An important difference compared to the RWGuard evaluation in Section 6.2 is that functional splitting in Cerberus considers only three functional group, that is: {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.

For the mimicry attack, 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 [29]. As before, in our VM evaluation Cerberus runs with 470 mimicry ransomware processes, which are all able to evade RWGuard, fully encrypting the VM files. This evaluation shows that our evasion techniques can generalize to classifiers trained on different datasets.

6.4 Evaluation against 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" [1], but rather leverages machine learning techniques [8]. 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.

For the mimicry attack, the mimicry behavior of Cerberus processes is modeled 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.

7 Countermeasures

Graph-based 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 [30] or unsupervised anomaly detection [20, 27]. 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 [20], 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 [19]. We note that both the functional splitting and mimicry attack can, by design, split operations in arbitrarily different ways. This would enable randomizing the attack behavior across different machines.

8 Related Work

Ransomware detection. For a review of behavioral ransomware detection techniques [1, 9, 12, 24, 29, 38], the reader is referred to Section 2. Other proposals [28, 35] focus specifically on entropy of written data to identify encrypted content. Depending exclusively on entropy is dangerous for reasons pointed out in Section 4.1. The use of *decoy files* has also been proposed for ransomware detection [16, 31, 33]. Decoys are a promising strategy, but they raise usability concerns, and their evasion is outside the scope of our work. Finally, for a discussion of relevant graph-based detection approaches [20, 27, 30] see Section 7.

Multiprocessing malware. Several ransomware families use multi-processing. This happens for example in WannaCry and Petya [4, 6]. Encryption is still performed by one process, while the others perform non encryption-related auxiliary tasks.

The CERBER ransomware (not to be confused with our *Cerberus* prototype), despite its name, does not appear to perform multi-process encryption. Instead, it focuses on obfuscation of static payload features [2]. MalWASH [23] and its successor D-TIME [36] 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.

Evasion of ransomware detectors. The work closest in spirit to ours is the critical analysis of ransomware defenses by Genç et al. [15]. Their work is more limited in scope than ours, considers a smaller set of features, and does not incorporate the notion of mimicry (focusing on simple feature obfuscation).

Adversarial sample generation. Generation of adversarial samples for various classes of malicious programs has been studied. This include generation of mobile [18] and conventional [10, 22, 37] malware binaries, and PDF-based file exploits [11, 13, 39, 40]. 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.Existing works [21, 37] aim at defeating malware detectors trained on dynamically-generated sequences of API calls. These proposals chiefly work by inserting dummy calls. Besides dummy calls, we also leverage a broader set of capabilities such as distributing calls across processes. This give our technique the ability to decrease per-process frequencies/counts of certain calls (necessary to defeat the detectors in our evaluation) without slowing down the attack, or to obfuscate data dependencies between calls (such dependencies are used by some detectors, e.g. [14]).

9 Conclusions

We demonstrated a novel practical attack against behavioral ransomware detectors. Our attack splits ransomware 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. However, the combined behavior of all the processes still successfully accomplishes a malicious goal.

We proposed three attacks, *process splitting*, *functional splitting*, and *mimicry*. Evaluation shows that our methods evade state-of-the-art detectors without limiting the capabilities of ransomware. To the best of our knowledge, this is the first comprehensive evaluation of this attack model in the ransomware domain.

References

1. Introducing the malwarebytes anti-ransomware beta. https://blog.malwarebytes.com/malwarebytes-news/2016/01/introducing-the-malwarebytes-anti-ransomware-beta/ (2016)

2. Cerber starts evading machine learning. https://blog.trendmicro.com/trendlabs-security-intelligence/cerber-starts-evading-machine-learning/ (2017)
3. I/o request packets. https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/i-o-request-packets (2017)
4. "Petya-like" Ransomware Analysis. https://www.nyotron.com/wp-content/uploads/2017/06/NARC-Report-Petya-like-062017-for-Web.pdf (Jun 2017)
5. 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)
6. Wannacry analysis and cracking. https://medium.com/@codingkarma/wannacry-analysis-and-cracking-6175b8cd47d4 (Nov 2018)
7. 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)
8. Malwarebytes anti-ransomware for business. https://www.malwarebytes.com/business/solutions/ransomware/ (2019)
9. Amin Kharraz, Sajjad Arshad, Collin Mulliner, William Robertson, Engin Kirda: UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In: USENIX Security Symposium (2016)
10. Anderson, H.S., Kharkar, A., Filar, B.: Evading Machine Learning Malware Detection p. 6 (2017)
11. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: ECML PKDD (2013)
12. Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., Maggi, F.: Shieldfs: A self-healing, ransomware-aware filesystem. In: ACSAC (2016)
13. Dang, H., Huang, Y., Chang, E.C.: Evading Classifiers by Morphing in the Dark. In: CCS (2017)
14. Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., Yan, X.: Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors. In: IEEE S&P (2010)
15. Genç, Z.A., Lenzini, G., Ryan, P.Y.A.: Next Generation Cryptographic Ransomware. In: Gruschka, N. (ed.) Secure IT Systems, vol. 11252, pp. 385–401. Springer International Publishing, Cham (2018)
16. Genç, Z.A., Lenzini, G., Sgandurra, D.: On Deception-Based Protection Against Cryptographic Ransomware. In: DIMVA (2019)
17. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. Computing Research Repository ArXiv (2014)
18. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial examples for malware detection. In: ESORICS (2017)
19. Gu, G., Porras, P.A., Yegneswaran, V., Fong, M.W., Lee, W.: BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In: USENIX (2007)
20. Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M.: Babar: Runtime Provenance-Based Detector for Advanced Persistent Threats. In: NDSS (2020)
21. Hu, W., Tan, Y.: Black-Box Attacks against RNN based Malware Detection Algorithms. arXiv:1705.08131 [cs] (May 2017), http://arxiv.org/abs/1705.08131
22. Hu, W., Tan, Y.: Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. arXiv:1702.05983 [cs] (Feb 2017), http://arxiv.org/abs/1702.05983, arXiv: 1702.05983

23. Ispoglou, K.K., Payer, M.: malwash: Washing malware to evade dynamic analysis. In: USENIX WOOT (2016)
24. Kharraz, A., Kirda, E.: Redemption: Real-time protection against ransomware at end-hosts. In: RAID (2017)
25. Kirda, E.: Unveil: A large-scale, automated approach to detecting ransomware (keynote). In: SANER (2017)
26. Lin, J.: Divergence measures based on the shannon entropy. IEEE Transactions on Information Theory (1991)
27. Manzoor, E., Milajerdi, S.M., Akoglu, L.: Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In: KDD (2016)
28. Mbol, F., Robert, J.M., Sadighian, A.: An Efficient Approach to Detect TorrentLocker Ransomware in Computer Systems. In: Foresti, S., Persiano, G. (eds.) Cryptology and Network Security, vol. 10052, pp. 532–541. Springer International Publishing, Cham (2016)
29. Mehnaz, S., Mudgerikar, A., Bertino, E.: Rwguard: A real-time detection system against cryptographic ransomware. In: Research in Attacks, Intrusions, and Defenses. RAID '18 (2018)
30. Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V.: HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In: IEEE S&P (2019)
31. Moore, C.: Detecting Ransomware with Honeypot Techniques. In: CCC (2016)
32. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: ACSAC (2007)
33. Moussaileb, R., Bouget, B., Palisse, A., Le Bouder, H., Cuppens, N., Lanet, J.L.: Ransomware's Early Mitigation Mechanisms. In: ARES (2018)
34. OKane, P., Sezer, S., McLaughlin, K.: Obfuscation: The hidden malware. In: IEEE S&P (2011)
35. Palisse, A., Durand, A., Le Bouder, H., Le Guernic, C., Lanet, J.L.: Data Aware Defense (DaD): Towards a Generic and Practical Ransomware Countermeasure. In: Lipmaa, H., Mitrokotsa, A., Matulevičius, R. (eds.) Secure IT Systems, vol. 10674, pp. 192–208. Springer International Publishing, Cham (2017)
36. Pavithran, J., Patnaik, M., Rebeiro, C.: D-time: Distributed threadless independent malware execution for runtime obfuscation. In: USENIX WOOT (2019)
37. Rosenberg, I., Shabtai, A., Rokach, L., Elovici, Y.: Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers. In: RAID (2018)
38. Scaife, N., Carter, H., Traynor, P., Butler, K.R.B.: CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In: ICDCS (2016)
39. Šrndic, N., Laskov, P.: Practical evasion of a learning-based classifier: A case study. In: IEEE S&P (2014)
40. Xu, W., Qi, Y., Evans, D.: Automatically evading classifiers: a case study on pdf malware classifiers. In: Networks and Distributed Systems Symposium (2016)