On the Effectiveness of Behavior-Based Ransomware Detection

Jaehyun Han[1(✉)], Zhiqiang Lin[2], and Donald E. Porter[1]

[1] The University of North Carolina at Chapel Hill, Chapel Hill, USA
{jaehyun,porter}@cs.unc.edu
[2] The Ohio State University, Columbus, USA
zlin@cse.ohio-state.edu

Abstract. Ransomware has been a growing threat to end-users in the past few years. In response, there is also a burgeoning market for anti-ransomware defense products, as well as research prototypes that explore more advanced, behavioral analyses. Intuitively, ransomware should be amenable to identification through behavioral analysis, since ransomware recursively walks a user's files and encrypts them, overwriting or deleting the plaintext. This paper contributes a study of the effectiveness of these behavior-based ransomware defenses, from both commercial products and academic proposals. We drive the study with a dead simple ransomware, augmented with a number of both straightforward and new evasion techniques. Surprisingly, our results indicate that most commercial products are strikingly ineffective. Ten out of 15 commercial products could not detect our simple ransomware without any evasive techniques; most of the rest were evaded and able to ransom user data with some combination of simple techniques. Only one tool appears to correctly identify our ransomware, but suffers from staggering false positives, including flagging Windows Explorer, Firefox, and Notepad as ransomware during routine operation. Our paper identifies a number of techniques to manipulate entropy to match the original file. The paper further shows that partial encryption, of as little as 3–5% of a file's data is sufficient to ransom most file formats. Finally, we show that a combination of these techniques can render an aggregate malice score that is well below that of a Linux kernel compile. In summary, these results indicate that it is highly likely that ransomware will be able to adapt its behavior to fit within the range of expected benign behaviors, avoiding detection even by future generations of behavioral ransomware detectors.

Keywords: Ransomware · Malware

1 Introduction

Ransomware is a growing threat for computer users, especially smaller businesses and less savvy users. For instance, over 16 million US dollars of bitcoins have been paid in exchange for ransom from roughly 19,750 victims in the years

N. Park et al. (Eds.): SecureComm 2020, LNICST 336, pp. 120–140, 2020.
https://doi.org/10.1007/978-3-030-63095-9_7

2016 and 2017 [13]. In a nutshell, ransomware renders a user's data unavailable, typically by encryption, and then charges the data owner a ransom to recover the data. Encryption is preferred to exfiltrating data, as the attacker need not store or return the ransomed data, but, rather, can simply sell a decryption key to the victim. This strategy is adopted by high-profile ransomware, including CryptoLocker [28], Cerber, and WannaCry [21]. In principle, ransomware should be a non-issue when users and enterprises follow best practices with respect to back-ups and least administrative privilege, as data can be restored from a secured back-up, nullifying the ransomware's leverage over users. Unfortunately, many users and businesses often do not follow these best practices, fall victim to ransomware, and pay the ransom, because the loss of essential data, such as patient or billing records, is more costly than the ransom.

The rising concern about ransomware has led to both commercial products, such as CyberSight RansomStopper, Acronis Ransomware Protection, Check-MAL AppCheck, CryptoDrop [30], and ZoneAlarm Anti-Ransomware as well as research prototypes, including Redemption [16], ShieldFS [5], and RWGuard [20], that purport to detect ransomware, in the same vein as inexpensive commercial virus scanners. Early detection has obvious benefits, primarily that ransomware can be stopped before much data is lost. Because the behavior of ransomware follows a fairly straightforward pattern, namely traversing the file system and encrypting data, there is a basis for optimism that a behavioral malware detector could be effective against ransomware. Indeed, most of the products and prototypes listed above use behavioral detection.

This paper studies the efficacy of these ransomware detectors and their underlying strategies. Although these commercial detectors are generally closed source, we develop a simple ransomware, a python script in less than 100 lines of code, and vary its behavior to infer what these detectors are monitoring. We identify several key features that these detectors use, including file system behavior monitoring and decoy file monitoring. Recent research prototypes have proposed to augment these features with monitoring for changes in file entropy [5,15,16,20,30]. In most designs, systems combine these features. For instance, Redemption [16] calculates a weighted average of these features, or a *malice score*. For each of these principal features, the paper then explores, through more targeted experiments, the degree to which these features can be manipulated or evaded by a more sophisticated ransomware.

In short, our results indicate that, counter-intuitively, the behavioral approach to ransomware detection is fragile in practice, and highly unlikely to work against a sophisticated adversary. First, we find that **several commercial products cannot detect our "textbook" ransomware** (Sect. 4.1). Second, we consider the individual behaviors that are monitored and combined to form a malice score. We demonstrate techniques that can effectively ransom users' data, while staying within a range that is indistinguishable from benign application behavior. Specifically, this paper investigates:

– **Entropy.** (Sects. 3.1 and 3.2) By definition, a good encryption function should yield a high-entropy ciphertext. A number of research prototypes look

for a shift in entropy, either in reads versus writes or in original versus overwritten file contents, as an indicator of the presence of ransomware. First, we observe that many common file formats have high entropy (e.g., `pdf`, `jpg`), and encrypting these files does not shift their entropy outside of the expected range. Second, for low-entropy files, this paper introduces several techniques to manipulate entropy that still deprive users of their data. For instance, low entropy files are necessarily compressible; if the ransomware first compresses the file, encrypts the compressed contents, and then pads the resulting file to its original length with regular contents, the encrypted output file's entropy will be comparable to the original.

– **File Overwrite or Deletion Rate.** (Sect. 3.3) Another natural monitoring strategy is to identify processes that delete a large swath of files (in the case where the encrypted versions are written elsewhere), or that overwrite a large number of files in place (with the encrypted version). In the case of monitoring for file overwrites, we show that one can evade this detection mechanism by only encrypting portions of a file. For most file formats, there is important metadata that can be encrypted, and that renders the entire file useless, at least for the average user. We show that these techniques are not easily undone by free or inexpensive file recovery tools. Although one might be able to pay an expert to reconstruct this metadata, these costs are often commensurate with the ransom.
– **Decoy Files.** (Sect. 3.4) Another common strategy for ransomware detection is to place decoy files on the users' file system, and monitor whether those specific files are deleted or overwritten. We show that, in practice, the naming and placement of these decoy files is easy to predict and differentiate from the user's "real" data; thus, it is trivial for ransomware to simply avoid these decoy files.

In summary, this paper demonstrates considerable cause for pessimism about the behavioral approach to ransomware protection. Although it is possible that there is room for improvement in behavioral analysis, the margin where it can identify malware without excessive false positives is likely narrow. We do note that backups are not a panacea, as backups without security isolation can themselves be ransomed. On balance, we find that end users would be better served to spend their IT budget on incremental backups within a separate administrative domain and, more generally, securing their infrastructure, than on ransomware-specific products.

2 Background

This paper focuses on ransomware that holds a user's *data* hostage, in order to extort payment from the user. There are other types of ransomware that are either scams, such as misleading the user to believe they have a virus and should pay to remove the virus [17], or that ransom the *system*, such as by locking the bootloader, where the data itself is still available on the file system [29]. In this

paper, we focus on ransoming encrypted data, as this is a growing threat and harder for users to recover from [29].

The simple problem is that when users pay to recover their data, generally out of desperation for business-critical data (e.g., billing records) or sentimental data that was not backed up (e.g., baby photos), there is no guarantee they will be able to recover their data. A 2018 report [7] states that more than half of victims who paid a ransom failed to recover their data. There are projects that can decrypt ransomed data for victims of well-known ransomware [23], but these typically rely on flaws in the use of cryptography, such as reuse of a common encryption key; if ransomware makes proper use of cryptography, there is no reason to believe that any reasonable amount of analysis on the ransomware binary or source code will lead to a decryption tool. Thus, research in this space focuses on identifying ransomware attacks as they are in progress, and stopping the ransomware before the data is lost.

As with most malware, we adopt a common assumption that an attacker will be able to install and execute the ransomware. System defenses are not foolproof, and users are prone to exploitable behavior, such as downloading code from the internet that includes malware [2]. Similarly, anti-virus software can scan for known static features of ransomware. Such as signatures of ransomware binaries, encrypted file extensions or static ransom notes. We assume a strong adversary that is evolving the ransomware over time. For instance, the Cerber ransomware generates a new binary with a new signature every 15 s [22].

Based on a mixture of documentation, papers, and our own experiments, Table 1 summarizes the principal features and techniques used by a range of ransomware detectors, namely: file entropy (Sect. 2.1), file system operations (Sect. 2.2), or decoy files (Sect. 2.3). This section explains how each feature is used in greater detail.

Table 1. Detection methods used by state-of-the-art ransomware detection systems

	Data entropy	File system operations	Decoy files
Research prototypes			
Redemption [16]	✓	✓	
ShieldFS [5]	✓	✓	
RWGuard [20]	✓	✓	✓
Commercial products			
CryptoDrop [30]	✓[a]	✓	
CyberSight RansomStopper[b]		✓	✓
Acronis Ransomware Protection[b]		✓	
CheckMAL AppCheck[b]		✓	
ZoneAlarm Anti-Ransomware[b]		✓	✓

[a]CryptoDrop claims to use entropy to detect ransomware. However, we couldn't find evidence that they use entropy in the distributed version.
[b]These are closed-source software and do not document their methods; we base this table on monitoring their behavior.

2.1 Data Entropy

File entropy [31] is a measurement, typically from 0..1, of how uniformly distributed the byte values in a file are. At one extreme, when a content of a file consists of the same byte value, the entropy of the file is 0. At the other, a file with a uniformly random distribution of byte values should have an entropy approaching 1. In principle, any strong encryption algorithm should have high-entropy outputs.

To give a sense of expected entropy values for common file types, we measure the entropy of a corpus of 240 document files, shown in Fig. 1 and grouped by type. We selected the first 30 files of each file type from the Govdocs1 corpus [11]. As a result, we collected 180 Microsoft Office documents, 30 pdf documents, and 30 jpeg image files. We can see that the entropy is widely distributed. Legacy office formats (`.doc` and `.xls`) have the lowest entropy. Most media formats, as well as Microsoft's Office Open XML documents (e.g., `docx`, `pptx`, `xlsx`) use compression, which leads to high entropy.

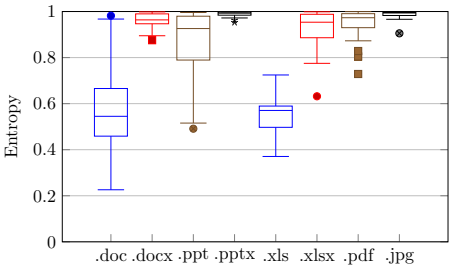


Fig. 1. Entropy distribution of a corpus of 240 document files, grouped by common formats.

Since ransomware encrypts files, state-of-the-art ransomware detectors use data entropy as a feature to identify encrypted files by monitoring the entropy. Entropy is typically used in combination with other features, discussed in the following subsections. Some detectors just monitor changes in the entropy of a given file [20]; others, such as Redemption [16] compare the entropy of reads and writes from a given process. In either approach, the detector looks for a significant upward shift in entropy, which would indicate that a file is likely being encrypted. In order to avoid excessive false positives, this approach necessarily involves some sensitivity analysis to "normal" and "abnormal" increases in entropy. This also relies on a flawed assumption that there is room for entropy to increase—i.e., that the file format is not already effectively at entropy 1, as is common for formats such as `.jpg` and `.pptx`. In other words, the efficacy of this feature rests on the assumption that one cannot effectively ransom a file's contents via encryption without significantly raising entropy.

2.2 File Overwrite and Deletion

Another common feature monitored by ransomware detectors is file overwrite and deletion. Intuitively, if one wants to ransom a file by encrypting the contents, the original contents must be overwritten with ciphertext, or, if the ciphertext is in a different location, the original file must be deleted or renamed over.

As with entropy, this monitoring requires some sensitivity to differentiate normal and abnormal behavior. Programs routinely update a portion of a file,

or even rewrite an entire file. Speaking generally, we have observed a few common measurement strategies. First, one can measure the percent of bytes overwritten versus the total size of the file, called the *overwrite ratio*. Second, one can monitor the total number of writes or deletions. Third, one can monitor the frequency of write or delete operations. If one of these values rises above a certain threshold, the process is flagged as malicious. Thus, for this feature, the goal for ransomware is to stay below this threshold, but still, deprive the victim of their data.

A related strategy considers the file type or extension written. Benign programs usually write a small number of specific file types only. For example, Microsoft Word will mostly write `*.doc` or `*.docx` files. The detector will flag as potentially malicious a process that writes to multiple different types of files.

Directory Traversal. Some ransomware detectors also monitor for recursive directory scans, typically in concert with other write operations. Because recursive directory scans are also executed by a number of benign applications, including backup utilities and virus scanners, it is easy for this feature to create false positives that irritate users and erode faith in the tool. This paper does not consider directory traversal in great detail, except to show in Sect. 4.5 that spreading this work to a separate process than the encryption work is sufficient to avoid detection by the commercial detectors that use this feature. Moreover, Sect. 5 shows that benign applications, such as git, can easily skew composite metrics based on this behavior.

2.3 Decoy Files

Ransomware detectors may also create a set of decoy files with various file types in the file system and monitor any changes to those files. Ransomware tends to encrypt all documents in the system. Thus when the ransomware tries to encrypt decoy files, the ransomware detector will notice the change in the expected contents of the file, and flag the writing process as ransomware.

2.4 Combining Techniques

As illustrated in Table 1, most ransomware defenses calculate a weighted average of a subset of the above techniques to form a global score, sometimes called a *malice score*. Redemption [16] calculates a malice score using six features. These six features are Entropy Ratio of Data Blocks, File Content Overwrite, Delete Operation, Directory Traversal, Write access to different type of files, and Access Frequency. Redemption scores each of the six features using different formulas, and then uses a weighted average of the six individual scores; if this average is above a threshold, the process is classified as ransomware. In general, combining features can lead to more robust classification, although setting appropriate weights can be a challenge.

3 Avoiding Detection

Section 2 explained the various features that are commonly used by ransomware detectors, typically as a weighted average or score, to flag processes as ransomware. This section explains how each of these mechanisms can be circumvented. We note that each of the avoidance techniques in this section also has a behavior that could be monitored; these behaviors overlap with common, benign patterns, and the heart of the question is whether there are behaviors that clearly delineate ransomware from benign software. Our results indicate that the line between ransomware and benign software is finer than one might expect.

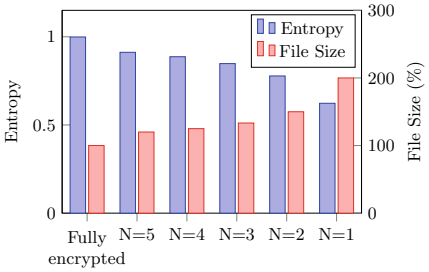


Fig. 2. Entropy and size of an encrypted file with a null, padding byte interleaved after every N bytes of ciphertext. Fully encrypted has no padding bytes interleaved. File size is measured as a percent, relative to the original plaintext. There is a smooth trade-off between size and target entropy, while still withholding the user's data. (Color figure online)

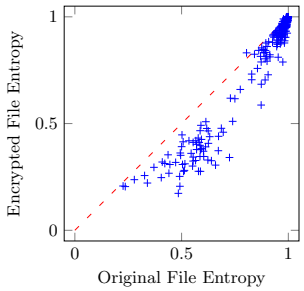


Fig. 3. A comparison of the original and encrypted entropy when using the compress-encrypt-pad strategy, using a corpus of 240 sample files. Points on the $y = x$ line represent an encrypted file with the same entropy and size as the unencrypted file; points below this line represent a reduction in entropy with the same size; points above the line represent an increase in entropy for the same size.

3.1 Entropy Laundering

Monitoring for a shift in entropy has an intuitive appeal, as encrypted outputs are necessarily going to be high entropy. This monitoring can either detect shifts in the contents of specific files, or in the difference between read and written data from a process.

As established in Fig. 1, one fundamental challenge is dealing with high-entropy file formats, such as a compressed Office document. The issue is that these files already have entropy close to 1, and normal file edits can increase small the entropy by small amounts—even as high as 1. In other words, for these file formats, entropy monitoring will have an obnoxiously high false positive rate and is simply not a good feature to monitor in these cases.

Thus, this section focuses on the efficacy of monitoring lower-entropy file formats. In other words, can we encrypt the file contents without changing the overall entropy of the file? Our basic strategy, which we will iteratively refine, is simple: we can encrypt the file (in memory) and then lower the entropy before writing it to disk by appending the same byte value (say zero) to the end of the file (let's call these the *padding bytes*). In practice, a more sophisticated attacker might instead interleave the padding bytes and the ciphertext.

The resulting entropy for interleaving a null padding byte between every N bytes of ciphertext is shown in Fig. 2. *Fully encrypted* shows the entropy and size of an encrypted file with no padding. The file used here is a PowerPoint slide (`.ppt`) and entropy of the file is 0.55 and the file size is 1.9 MB. We encrypt the file using `AES` algorithm in `CBC` mode.

Since entropies of encrypted files are mostly near one, this result is independent of the original file's entropy. When N = 1, this means the resulting file alternates between a byte of ciphertext and a byte of padding, yielding an entropy of about 0.6, or at roughly the first quartile of the lowest entropy formats (e.g., `.doc` and `.xls`).

The second, red bar in the figure represents size. There is a fairly smooth curve from N = 5..N = 1 in terms of trading size for lower entropy, and, although not pictured, the trend can be extended beyond N = 1 if needed. If changing the size is no issue, this experiment demonstrates that it is straightforward to ensure an encrypted output with comparable entropy to the input.

Of course, ransomware detectors can also monitor for changes in a file size or file overwrites, which we address in the next two subsections, respectively.

3.2 Maintaining File Size

Because many ransomware detectors also factor in file system operations, growth in a file's size is noticible; we refine the entropy laundering technique to preserve a target file size. We remind the reader that entropy monitoring is only effective for low entropy file formats.

Our second, simple observation is that low entropy files are highly compressible. By using this characteristic, ransomware can first compress then encrypt the smaller data payload, and finally, interleave the contents with padding to lower the entropy and yield an output that matches the original file size.

To evaluate this, we first compress a file using the `zlib` module in Python. Then we encrypt the compressed contents using the `AES` algorithm, and then pad the resulting output to match the original file size. Figure 3 compares the original file entropy (on the x-axis) to the encrypted file entropy (on the y-axis). A perfect match in entropy would be a $y = x$ line. The resulting distribution is almost entirely *at or below* $y = x$; i.e., this process often yields a *lower* entropy than the original file, and, in the worst case, the entropy is only raised by 3.4%.

This experiment shows that entropy can be lowered while maintaining the file size, but at the cost of overwriting the entire file. Overwriting the file (or deleting the file after writing these files elsewhere), is still easily detected by several ransomware detectors.

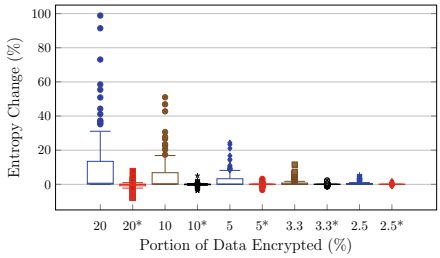


Fig. 4. Entropy changes in percentage value for the partially encrypted file compared to the original file for a corpus of 240 files. We encrypt a portion of the data from 20–2.5%. Star (*) represents that we used the compress-encrypt-pad strategy. The graph shows that the entropy changes are small, even when 20% of the file data is encrypted.

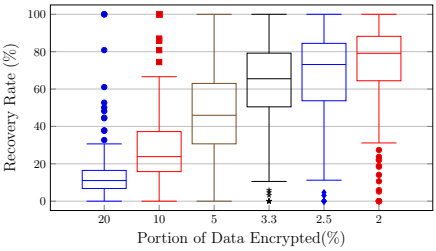


Fig. 5. Recovery rates for partial encryption over a corpus of 200 pdf files, when a portion of the data from 20–2% is encrypted. Block size of 256 bytes is used. Lower is better. The results indicate that, even if only 5–20% of the file is encrypted, roughly half of the data is unrecoverable.

3.3 Partial Encryption

The goal of ransomware is to deny users their file contents. The techniques presented so far accomplish this goal at the cost of overwriting an entire file, which can be detected by common techniques. This Subsection presents a technique that only overwrites a portion of a file, but effectively denies the user access to their data.

We observe that most file formats are brittle, with essential, non-redundant metadata or other data whose interpretation is predicated on the previous bytes. Corrupting (or encrypting) a relatively small portion of the file may be enough to render the file unusable to most end-users. One can thus mimic small updates to a file from a legitimate program with a combination of small updates to critical portions of the file, with the techniques above that can preserve the same average entropy of the encrypted bytes within the same space.

We first note a caveat to this attack: an expert in a given file format may be able to recover part of a file's contents. Specifically, the expert may successfully recover the data structure of the file format; encrypted data cannot be recovered. For instance, an engineer on the Adobe Acrobat team might be able to recover part of a partially encrypted pdf document by hand. We expect that, even if an end-user could find such an engineer, the hourly rate to recover a large data set would quickly approach the cost of the ransom (typically on the order of hundreds for individuals or tens of thousands of dollars for enterprises [9]). Thus, to evaluate the efficacy of this technique, we primarily consider automated file repair tools that are free or inexpensive (hundreds of dollars).

We define the success of this attack as: (1) updating only a small portion of the file (less than 20%), (2) preserving the same size (3) preserving comparable

entropy, and (4) rendering most files unopenable and unrepairable by free or inexpensive tools.

As a simple proof-of-concept, we encrypted every N-th block of data in the file to measure its effect on entropy. For instance, when encrypting 20% of the file's data, we encrypted every 5th block. We used the AES algorithm with a block size of 1024 bytes to encrypt data. We also used compress-encrypt-pad for the target portion of data to see the entropy changes.

This is an under-approximation of a more targeted encryption, but allows us to measure the efficacy of the overall approach. Figure 4 shows the entropy changes in percentage value. The entropy of a file is always increased because we still encrypt a part of the file. However, when we use the compress-encrypt-pad strategy described in Sect. 3.2, the maximum entropy change is less than 8% when encrypting 20% of the file data. Most papers do not specify a particular threshold for entropy, except RWGuard, which uses 6 (or 0.75 on our scale from 0..1). In this experiment, entropy shift is at most 0.07, even at an aggressive 20% of encryption. A lower threshold would incur false positives, so we conclude that this change is unlikely to trigger detection.

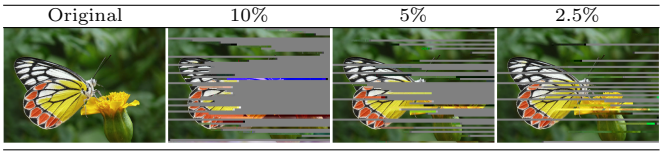
To measure whether partial encryption is effective at withholding user data, we collected 200 different PDF documents from the web using Common Crawl Document Download [4]. We choose PDF documents with a minimum of 10 pages. We partially encrypted these 200 files and recovered encrypted files using the `pdrepair` utility from pdf-tools [25]. In practice, many documents are partially recoverable or corrupted. In order to conservatively quantify the amount of the document that can be recovered, we measure and compare the ink coverage of the original and recovered documents using `Ghostscript`. Ink coverage is a fraction of paper that is covered in each CMYK ink color.

Figure 5 shows the recovery rate distribution while the amount of encrypted data changes. We used a block size of 256 bytes in this experiment. The recovery rates increase as the amount of encrypted data decreases. We can see the average recovery rate is about 15% when 20% of the file is encrypted. When we encrypted 5% of data, recovery rate increases. However, the average recovery rate is still about 48%. At even lower percentages (3.3% and below), more than 64% of data is recoverable. Thus, we see a "sweet spot" when 5–20% of the file is encrypted, roughly half of the user's data is withheld.

As another point of comparison, and for more visual intuition about the data recovery, we did the same experiments with a JPEG image. The sample encrypted JPEG images are shown in Fig. 6. In this experiment, we change the percent of the file that is encrypted using a 256 bytes block size.

When we encrypt 10% of image data, it is difficult to recognize the image contents. However, when we encrypt 2.5% of image data, an encrypted image is annoying but we can figure out that the original image contains a butterfly and a flower. Overall, encrypting a modest portion of the file (5% or more) is effective at rendering the photograph useless to the average user.

We try to recover the partially encrypted images from the first experiment (Fig. 6a) using Adobe Photoshop. We recover the damaged areas using content-



(a) Sample JPEG image after encrypting a portion of the data, from 10–2.5%.

(b) Recovery of partially encrypted images from the first row, using Photoshop.

Fig. 6. Sample images resulting from various partial encryption parameter settings.

aware fill feature [8]. This feature fills a selected area in an image using information from the rest of the image. For all three images, Photoshop generates reasonable images, shown in Fig. 6b. However, Photoshop couldn't recover the details because it doesn't have any information about the damaged area, rather, the encrypted regions appear more blurred together to mask some of the most noticeable artifacts. Nonetheless, for photos with sentimental value, such a baby photos, we expect this level of recovery would be unacceptable and many users would pay a ransom to recover high-quality images.

In summary, the partial encryption technique can avoid detection by only writing to a small portion of a file (effective at 3–5% of total bytes overwritten, which is commensurate with light edits to the file in a legitimate application), preserving the same file size, and rendering the majority of the data unrecoverable for less than the cost of the ransom.

3.4 Decoy Avoidance

Some ransomware detectors generate decoy files in a file system. They monitor these specific files and flag the process when these files are modified. This subsection analyzes decoy files and their generation pattern.

We examine the decoy files generated by three commercial products, CyberSight RansomStopper, Cybereason RansomFree, and ZoneAlarm Anti-Ransomware. All three products create a decoy directory to store decoy files in important user directories, such as "`My Documents`" or "`Desktop`". Thus decoy files are not necessarily placed with other user files. In the decoy directory, various types of document and multimedia files are generated as decoy files, because some ransomware encrypts only specific file types. All three ransomware detectors generate one of each file type in the directory. So we start by creating two handwritten rules to detect a decoy directory: if more than three different types

of files exist in a directory, and no more than two files of the same type exist in the directory, we conclude that it is a decoy directory. If a directory meets these two conditions, ransomware can just avoid this directory. In short, these decoys in practice are painfully obvious and easy to avoid.

To simulate a more sophisticated ransomware detector, we flatten this directory structure and collect 350 filenames. Filenames of decoy files are generated as a combination of random words. Some detector put a message to not delete the file in every filename, such as "`Endpoint_Resume_Do NotRemove.doc`". We use collected filenames to train a Naive Bayes classifier. Our classifier can classify files with 88.5% accuracy and 98.3% recall. High recall means a ransomware can detect most of the decoy files with very low false negatives.

Of course, more sophisticated decoy file generation is possible. The challenge is creating files that are truly indistinguishable from a user's data, and it is likely that efforts to improve decoy creation would lead to an "arms race" between decoy detection and decoy generation. The more decoy files look like user files, the harder it will be for ransomware to avoid; on the other hand, this also improves the risk of the user deleting a decoy or editing the decoy by accident, and triggers a false alarm.

3.5 I/O Rate: Slow, Multi-process Attack

In monitoring file system operations, ransomware detectors often factor in the rate of I/O. As a result, recent ransomware is slowing down the encryption process to avoid detection [27].

Similarly, we found that most ransomware detectors are sensitive to spreading the work across multiple processes. Sometimes anti-ransomware monitors a behavior of individual processes. An attacker spreads out the attack on multiple processes. Each process performs small operations and is not suspicious to detector [26].

4 Commercial Products

We evaluated commercial anti-ransomware products, 9 anti-ransomware products: CyberSight RansomStopper (3.1.1), Acronis Ransomware Protection (Build 1700), CheckMAL AppCheck (2.5.35.2), CryptoDrop (1.5.353.1336), ZoneAlarm Anti-Ransomware (1.1.1023.17955), Cybereason RansomFree (2.4.2.0), Bitdefender Anti-ransomware (1.0.12.151), Malwarebytes (3.6.1.2711), and Trend Micro Ransom Buster (12.0.1150) and 6 anti-virus products: Webroot Secure Anywhere (9.0.24.49), Kaspersky Anti-Virus (1.1.534.17681), ESET NOD32 (11.2.49.0), AVG AntiVirus (18.6.3066), Avast Free Antivirus (18.6.2349), and McAfee Total Protection (16.0 R12). We select products that specifically advertise ransomware protection. These software are closed-source, and thus we cannot confirm in all cases whether they monitor behavioral features; that said, some products do specify that they monitor behavioral features (e.g., file activities) in their datasheet or website.

We first test the ability of these products to detect known, real-world ransomware samples. Five ransomware samples are used in this experiment: TeslaCrypt, Jigsaw, Locky, Cerber, and WannaCry. These ransomware families infected a large number of victims and were released on or before 2017 [13,34]. Therefore, we expect they are well-known to the anti-malware industry. All five samples encrypt and ransom user files. We execute these samples on a system running each anti-ransomware product. Except for Bitdefender Anti-Ransomware and Webroot Secure Anywhere, all commercial products can successfully detect all of these real-world ransomware samples. Bitdefender Anti-Ransomware is designed to detect only one of three ransomware families, CTB-Locker, Locky, and TeslaCrypt, but it could not detect any of the five samples. Webroot Secure Anywhere could detect TeslaCrypt and Jigsaw samples but it could not detect Locky, Cerber, and WannaCry samples. Four anti-virus products detect these samples right after the binary file is copied to the system, even before execution, namely: Kaspersky Anti-Virus, ESET NOD32, AVG AntiVirus, and Avast Free Antivirus. This result shows most of the target products are effective for known ransomware.

Second, we evaluate commercial products using a simple, hand-written ransomware in python. Our ransomware recursively encrypts each file in a user's home directory as quickly as possible. The ransomware encrypts a file using the `AES` algorithm in `CBC` mode with Python Cryptography Toolkit (`pycrypto`). When encrypting a file, the ransomware overwrites the file with encrypted contents. Unless otherwise specified, the ransomware uses a block size of 1024 Bytes. We created a Windows 7 VM in Virtual Box and experimented with commercial products in the VM. We place 3,000 files with sizes ranging from 1 KB to 106 MB in the user's home directory. These files are the first 3,000 files from the Govdocs1 corpus [11]. We placed the first 1,000 files in the user's `My Documents` directory, the next 1,000 files in the user's `Downloads` directory, the last 1,000 files in the user's desktop. The total size of these files is 1.6 GB. When an anti-ransomware can't detect our ransomware, we ran the test at least two times until seeing the consistent results. We plan to release our ransomware scripts and supporting data upon publication. With this "toy" ransomware, we measure the effectiveness of each of the evasion techniques described in the previous section.

4.1 Basic Ransomware, No Evasion

The most striking result is that our textbook ransomware was not detected by four of the anti-ransomware products: Cybereason RansomFree, Bitdefender Antiransomware, Malwarebytes, and Trend Micro Ransom Buster nor by any of the 6 anti-virus products that advertise ransomware defense. We assume that the reason is that these products solely rely on static features, such as matching known binary signatures in order to detect ransomware. Our hand-written ransomware would not match a known signature. The most likely explanation is that these products are not behavior-based ransomware detectors. Therefore,

Table 2. Commercial anti-ransomware software, and their ability to detect stealthy ransomware. A checkmark in the table means anti-ransomware can detect the ransomware; a percentage indicates the threshold below which the detector could no longer detect the ransomware.

	Basic ransomware	Basic ransomware (New file)	Partial encryption	Decoy avoidance	Slow encryption	Multi-process encryption
Anti-ransomware						
CyberSight RansomStopper	✓	✓	✓		✓	*
Acronis Ransomware Protection	✓	✓	✗ (33%)	✓[a]		
CheckMAL AppCheck	✓	✓	✗ (25%)	✓[a]	✓	
CryptoDrop	✓	✓	✓	✓[a]	✓	✓
ZoneAlarm Anti-Ransomware	✓		✓	✓	✓	

[a]These anti-ransomware products don't use decoy files.

we excluded these products, and did the further analysis on 5 behavior-based ransomware detectors.

AVG AntiVirus has a feature called strict ransomware protection mode. With this mode on, it can detect ransomware but any process that writes to a specified directory is flagged. Consequently, the false positive rate is very high—even a benign word processor is required to get approval to save a document.

Some ransomware creates a new file with encrypted data and deletes the original. Thus, we create a variant of basic ransomware that creates encrypted versions and then deletes the user files. However, most commercial products we tested behave the same toward both variants of the basic ransomware. Only one anti-ransomware product, ZoneAlarm Anti-Ransomware, can't detect ransomware when it creates a new encrypted file and deletes the original.

Table 2 shows these anti-ransomware products and their effectiveness on evasion techniques, explained in more detail below. Anti-ransomware products which couldn't detect the basic ransomware are excluded. A checkmark in the table means that the ransomware detector can detect the ransomware with the feature. In partial encryption, the percentage value indicates the highest portion of encryption that the anti-ransomware cannot detect; a checkmark means the tool can detect any amount of encryption.

4.2 Partial Encryption

We implement partial encryption (Sect. 3.3) in our basic ransomware. RansomStopper, CryptoDrop, and ZoneAlarm Anti-Ransomware can detect the ransomware regardless of the portion of data encrypted. However, Acronis Ransomware Protection cannot detect when $N \geq 3$ which means the ransomware encrypts less than or equal to 33% of file data. AppCheck can't detect when $N \geq 4$ that the ransomware encrypts less than or equal to 25% of file data.As discussed in Sect. 3.3, a user will lose more than 75% of the contents of a file, when only 10% of the data in the file is encrypted.

4.3 Avoiding Decoy Files

Two of the products in Table 2 generate decoy files: CyberSight RansomStopper and ZoneAlarm Anti-Ransomware. Interestingly, these monitors will tolerate small changes to these decoy files without flagging the process as ransomware. Thus deleting or overwriting on one of the decoy files won't trigger the detector. This implies that these detectors monitor a combination of other features to make a decision. This makes some intuitive sense, as decoy files are user-visible and may be accidentally removed by a user.

We avoided encrypting decoy files using the hand-written rules described in Sect. 3.4. This was sufficient to avoid detection by CyberSight RansomStopper. ZoneAlarm can detect the ransomware even when we didn't modify any decoy files.

4.4 Slow Encryption

We measure the impact of encryption rate on the ransomware detectors. We modified the basic ransomware to wait 60 s between encrypting each file. To evaluate slow encryption, we used 300 target files instead of 3000 files. We choose the first 100 files in each of 3 directories. Since we waited 60 s per one file encryption, total encryption time was approximately 330 min, whereas the basic ransomware took less than 10 min to encrypt entire user files (3000 files). This change was sufficient to elide detection in Acronis Ransomware Protection.

4.5 Multi-process Encryption

To measure the sensitivity to dividing the work across processes, one process recursively traverses directories and then forks workers that encrypt a single file and then exit. This was sufficient to evade three of the ransomware detectors: Acronis Ransomware Protection, CheckMAL AppCheck, and ZoneAlarm Anti-Ransomware. A fourth detector, RansomStopper, shows an interesting result: it detects modifications on decoy files, but only kills the worker process, letting processes continue encrypting all of the other files.

4.6 False Positives on CryptoDrop

In the previous experiments, CryptoDrop shows strong performance on detection; it detected all of our evasion strategies. Upon further investigation, however, we find that CryptoDrop is simply monitoring heavy writes to multiple files, and tested whether it would also trigger high false positives. First, we open a new Windows Notepad and write around 1 KB text and save in `My Documents` directory. When we save copies of this file with 20–30 different names using the "`Save As...`" menu, CryptoDrop labels Windows Notepad as ransomware. Next, we extract a zip file which consists of 50 files in `My Documents` directory. We used 7zip and Windows Explorer to extract the file, and CryptoDrop flags both 7zip and Windows Explorer as ransomware. Finally, we browse the web

normally using Firefox. We downloaded a file to `My Documents` directory after every 3–5 pages. When we downloaded the 5th file, CryptoDrop labels Firefox as ransomware.

Although CryptoDrop can detect all of our evasion techniques, it comes at a cost of excessive false positives for users. This result is consistent with the overall hypothesis that there is a very slim range of behaviors that are unique to ransomware.

4.7 Entropy

Although several proposed ransomware detection methods use file entropy as a feature [5,15,16,20], we observed that most commercial ransomware detectors seemed insensitive to the entropy of a file write.

As a simple experiment, we modified our basic ransomware to simply overwrite 25% of file data with null bytes instead of encrypted bytes, which would lower the entropy. In this case, all of the commercial ransomware detectors flagged our ransomware based on the volume of data that was written. Although we cannot confirm that these detectors are not considering data entropy, the fact that they flag code as ransomware that issues large writes but lowers entropy indicates that these tools are likely insensitive to data entropy. It is possible that our ransomware was so simple that entropy detection was not triggered, and a more complex ransomware would trigger entropy monitoring; nonetheless, the observation that the entropy-lowering ransomware is detected is disquieting. We will return to the entropy experiment in Sect. 5, when we consider a state-of-the-art research prototype.

5 Research Prototypes

We contacted the authors of both Redemption [16] and ShieldFS [5]; neither provided us with a source or a binary drop. Thus, we instead did our best to reimplement the malice score as described in the Redemption paper.

To evaluate Redemption's malice score technique, we ran three benign applications and two variants of our basic ransomware: the non-evasive version, and the version that performs partial encryption (at 10% of the files' contents), and at a rate of one file per 2 s. 10% is selected as a relatively generous threshold; our experiments indicate we could easily drop to 3–5% if needed. The three benign applications are a git clone of Linux kernel, Linux kernel build, and bzip2 compression of Linux kernel code. This experiment is done on Linux, and we use `strace` to trace each process and score features using the trace.

We made two assumptions while scoring because some features are not clearly described in the Redemption paper [16]. First, when measuring the file overwrites, we didn't count newly created files as overwritten. Second, to score a directory traversal, we counted the maximum number of files that are accessed sequentially in the same directory; Redemption scores the "additive inverse of

the number of privileged accesses to unique files in a given path." The paper does not provide values for the thresholds.

Redemption calculates a malice score per process. All three benign applications invoke multiple processes. For computing the malice score, we merged I/Os from these processes, treating them as a single application.

The malice score over time is shown in Fig. 7. The highest malice score among the benign applications is a Linux kernel compile, which is consistently around 0.5. This is attributable to the build process frequently writing files in multiple directories. In Fig. 7a, `git clone` does not trigger a high malice score most of the time. Most of the time `git` downloads objects in a file without an extension. At the end of the cloning process, `git` creates a local branch from the `master` branch. At this moment, `git` creates a lot of source code files, with known extensions, in the file system. Consequently, the malice score increases to 0.61. Two features, high scores from directory traversal, and write access to different type of files contribute to git's spike in malice score. Finally, `bzip2` has a low malice score. Unlike other workloads, `bzip2` does not delete any files, writes to a single document class, and only traverses directories with read access. In total, we expect that these applications show the range of expected malice scores. In order to avoid heavy false positives, a ransomware detector would likely need to flag applications that spent significant time over 0.6.

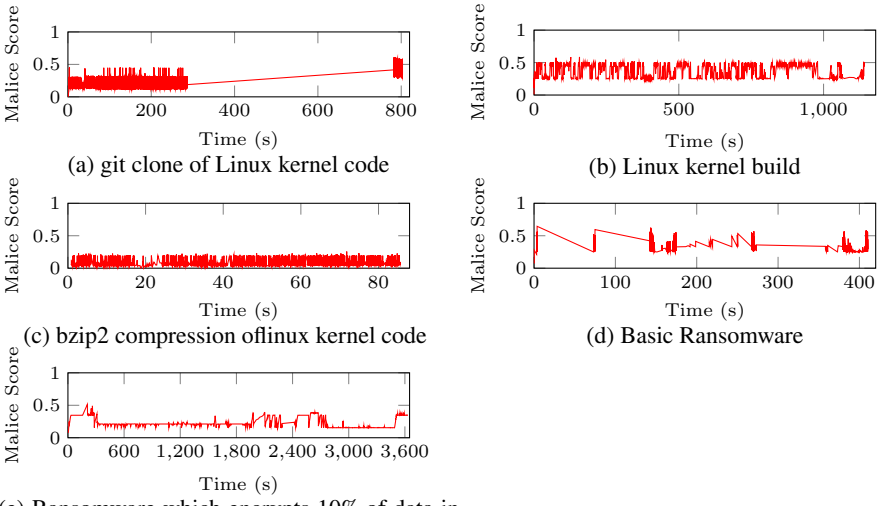


Fig. 7. Redemption [16]'s malice score over time of three benign application executions, a naive ransomware, and a more evasive ransomware. Lower is less likely to be malware. Our evasive ransomware has a consistently lower malice score than a Linux kernel compilation.

Results from our ransomware are in Fig. 7d and e. In the basic ransomware experiment (Fig. 7d), the malice score increases as high as 0.65, and would likely be flagged. This is primarily attributable to the directory traversal component. We observe that the malice score is more sensitive to directory traversal than the entropy ratio.

Partial and slow encryption lowers the malice score—usually at or below 0.4 and, at most, 0.51. Recall that anything below 0.6 is unlikely to be flagged as ransomware with this method.

6 Related Work

Similar to this work, Genç et al. studied ransomware detection and evasion techniques for key-oriented protection and behavioral analysis [12]. Similar to our work, the authors describe two evasion methods: pseudo-random permutation for encryption and partial encryption. Our work advances the state of the art in three ways. First, their proposed permutation method avoids entropy-based detection, but is less robust to reverse engineering by a victim than a standard cipher; our paper shows how to use a standard cipher with the compress-encrypt-pad method and still avoid using entropy-based detection. Second, the authors mentioned 20% encrypted files are unusable, but they didn't discuss the efficacy of the partial encryption. This paper shows that partial encryption is effective to extort user files even when a victim used the recovery tools. Finally, our paper studies additional evasion techniques, such as slow encryption, multi-process encryption, and decoy avoidance to evade more detecting methods.

One significant component of an attack, orthogonal to this paper, is loading malware onto the target system. Malware can get onto a computer by a user's explicit mistake, such as downloading an attachment from a phishing email or malicious website; more stealthily by using a system vulnerability [32]; or via a malware distribution service [2]. Most related to this paper, Gangwar et al. analyze these delivery methods and try to detect ransomware by delivery method [10].

Most ransomware needs to communicate with the ransomers who will collect the payment. Malware often has a command and control (C&C) server to control it remotely. In ransomware, C&C server is used to exchange encryption key or other client details. Several works are available to detect ransomware at the network level by monitoring and identifying these communications between the C&C server and the victim host [1,6]. Our toy ransomware uses a static key and does not perform any network communication. To evade detection by network activity some ransomware locally generates a key instead of getting one from a C&C server [24].

Some anti-ransomware create hooks in common cryptographic libraries. When ransomware tries to encrypt a file, the anti-ransomware can intercept and save the key. Then it decrypts the files using the saved decryption key. PayBreak [18] successfully retrieves the key from ransomware that uses Microsoft Cryptographic API [33]. The authors showed that it can even detect ransomware that statically links known cryptographic libraries, such as Crypto++.

Cheng et al. studied partial encryption methods on media files [3], in order to reduce encryption/decryption costs. Here, the goal is different—confidentiality of these files—whereas the goal of this paper is to explore the efficacy of partial encryption to ransom a user's data.

A system can implicitly provide a backup. Ransomware needs to overwrite the data in the hard drive. Modern hardware, such as Solid State Drives (SSDs), tend to leave the old files untouched because the device can only erase at the granularity of an erase block. FlashGuard [14] modifies SSD garbage collection to retain the copies of the old files for the file recovery when encrypted by ransomware.

Most general-purpose malware detectors such as signature-based are looking for ransomware. More precisely, for known ransomware. Marpaung et al. survey malware evasion techniques [19]. Since ransomware is a kind of malware, the attacker can use these methods to evade traditional malware detectors.

7 Conclusion

This paper demonstrates that there is considerable cause for pessimism about the effectiveness of host-level ransomware detectors. A "textbook" ransomware cannot be detected by a number of commercial anti-ransomware products; relatively straightforward evasive techniques can thwart the rest, yet still effectively ransom user's data. More sophisticated behavioral analyses are unlikely to fare better; a deeper exploration of each of the features proposed in the literature, as well as aggregate metrics, are unlikely to accurately distinguish ransomware from benign software.

Acknowledgments. We thank the anonymous reviewers, our shepherd Karim Elish, and Bhushan Jain for their insightful comments on previous drafts. This material is based upon work supported by the NSF/VMware Partnership on Software Defined Infrastructure (SDI) as a Foundation for Clean-Slate Computing Security (SDI-CSCS) program under Award "SDI-CSCS: Collaborative Research: S2OS: Enabling Infrastructure-wide Programmable Security with SDI", grant numbers CNS-1700512 and CNS-1834216.

References

1. Cabaj, K., Mazurczyk, W.: Using software-defined networking for ransomware mitigation: the case of CryptoWall. IEEE Netw. **30**(6), 14–20 (2016)
2. Caballero, J., Grier, C., Kreibich, C., Paxson, V.: Measuring pay-per-install: the commoditization of malware distribution. In: USENIX Security Symposium, p. 13 (2011)
3. Cheng, H., Li, X.: Partial encryption of compressed images and videos. IEEE Trans. Signal Process. **48**(8), 2439–2451 (2000)
4. Common Crawl Document Download. https://github.com/centic9/CommonCrawlDocumentDownload

5. Continella, A., et al.: ShieldFS: a self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 336–347. ACM (2016)
6. Cusack, G., Michel, O., Keller, E.: Machine learning-based detection of ransomware using SDN. In: Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, pp. 1–6. ACM (2018)
7. CyberEdge Group: 2018 cyberthreat defense report. Technical report, CyberEdge Group (2018)
8. Ding, M., Tong, R.F.: Content-aware copying and pasting in images. Visual Comput. **26**(6–8), 721–729 (2010)
9. Everett, C.: Ransomware: to pay or not to pay? Comput. Fraud Secur. **2016**(4), 8–12 (2016)
10. Gangwar, K., Mohanty, S., Mohapatra, A.K.: Analysis and detection of ransomware through its delivery methods. In: Panda, B., Sharma, S., Roy, N.R. (eds.) REDSET 2017. CCIS, vol. 799, pp. 353–362. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-8527-7_29
11. Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G.: Bringing science to digital forensics with standardized forensic corpora. Digit. Investig. **6**, S2–S11 (2009)
12. Genç, Z.A., Lenzini, G., Ryan, P.Y.A.: Next generation cryptographic ransomware. In: Gruschka, N. (ed.) NordSec 2018. LNCS, vol. 11252, pp. 385–401. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03638-6_24
13. Huang, D.Y., et al.: Tracking ransomware end-to-end. In: Tracking Ransomware End-to-end. IEEE (2018)
14. Huang, J., Xu, J., Xing, X., Liu, P., Qureshi, M.K.: FlashGuard: leveraging intrinsic flash properties to defend against encryption ransomware. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2231–2244. ACM (2017)
15. Kharraz, A., Arshad, S., Mulliner, C., Robertson, W.K., Kirda, E.: UNVEIL: a large-scale, automated approach to detecting ransomware. In: USENIX Security Symposium, pp. 757–772 (2016)
16. Kharraz, A., Kirda, E.: Redemption: real-time protection against ransomware at end-hosts. In: Dacier, M., Bailey, M., Polychronakis, M., Antonakakis, M. (eds.) RAID 2017. LNCS, vol. 10453, pp. 98–119. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66332-6_5
17. Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L., Kirda, E.: Cutting the gordian knot: a look under the hood of ransomware attacks. In: Almgren, M., Gulisano, V., Maggi, F. (eds.) DIMVA 2015. LNCS, vol. 9148, pp. 3–24. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20550-2_1
18. Kolodenker, E., Koch, W., Stringhini, G., Egele, M.: PayBreak: defense against cryptographic ransomware. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 599–611. ACM (2017)
19. Marpaung, J.A., Sain, M., Lee, H.J.: Survey on malware evasion techniques: state of the art and challenges. In: 2012 14th International Conference on Advanced Communication Technology (ICACT), pp. 744–749. IEEE (2012)
20. Mehnaz, S., Mudgerikar, A., Bertino, E.: RWGuard: a real-time detection system against cryptographic ransomware. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) RAID 2018. LNCS, vol. 11050, pp. 114–136. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00470-5_6
21. Mohurle, S., Patil, M.: A brief study of Wannacry threat: ransomware attack 2017. Int. J. Adv. Res. Comput. Sci. **8**(5) (2017)

22. Nieuwenhuizen, D.: A behavioural-based approach to ransomware detection. Whitepaper, MWR Labs Whitepaper (2017)
23. The No More Ransom Project. https://www.nomoreransom.org/en/index.html
24. Offline Ransomware Encrypts Your Data without C&C Communication. https://blog.checkpoint.com/2015/11/04/offline-ransomware-encrypts-your-data-without-cc-communication/
25. PDF Tools. http://www.pdf-tools.com
26. Ramilli, M., Bishop, M., Sun, S.: Multiprocess malware. In: 2011 6th International Conference on Malicious and Unwanted Software (MALWARE), pp. 8–13. IEEE (2011)
27. 11 ransomware trends for 2018. https://www.csoonline.com/article/3267544/ransomware/11-ways-ransomware-is-evolving.html
28. Richardson, R., North, M.: Ransomware: evolution, mitigation and prevention. Int. Manag. Rev. **13**(1), 10–21 (2017)
29. Savage, K., Coogan, P., Lau, H.: The Evolution of Ransomware. Symantec, Mountain View (2015)
30. Scaife, N., Carter, H., Traynor, P., Butler, K.R.: CryptoLock (and drop it): stopping ransomware attacks on user data. In: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), pp. 303–312. IEEE (2016)
31. Shannon, C.E.: The mathematical theory of communication (1963)
32. Snow, K.Z., Monrose, F., Davi, L., Dmitrienko, A., Liebchen, C., Sadeghi, A.R.: Just-in-time code reuse: on the effectiveness of fine-grained address space layout randomization. In: 2013 IEEE Symposium on Security and Privacy (SP), pp. 574–588. IEEE (2013)
33. Wiewall, E.: Secure your applications with the Microsoft CryptoAPI. Microsoft Dev. Netw. News **5**(96), 3 (1996)
34. Zimba, A., Chishimba, M.: Understanding the evolution of ransomware: paradigm shifts in attack structures. Int. J. Comput. Netw. Inf. Secur. **11**(1), 26–39 (2019)