Redemption: Real-time Protection Against Ransomware at End-Hosts

Amin Kharraz[1] and Engin Kirda[1]
{mkharraz,ek}@ccs.neu.edu

Northeastern University, Boston, USA

Abstract. Ransomware is a form of extortion-based attack that locks the victim's digital resources and requests money to release them. The recent resurgence of high-profile ransomware attacks, particularly in critical sectors such as the health care industry, has highlighted the pressing need for effective defenses. While users are always advised to have a reliable backup strategy, the growing number of paying victims in recent years suggests that an endpoint defense that is able to stop and recover from ransomware's destructive behavior is needed.
In this paper, we introduce Redemption, a novel defense that makes the operating system more resilient to ransomware attacks. Our approach requires minimal modification of the operating system to maintain a transparent buffer for all storage I/O. At the same time, our system monitors the I/O request patterns of applications on a per-process basis for signs of ransomware-like behavior. If I/O request patterns are observed that indicate possible ransomware activity, the offending processes can be terminated and the data restored.
Our evaluation demonstrates that Redemption can ensure zero data loss against current ransomware families without detracting from the user experience or inducing alarm fatigue. In addition, we show that Redemption incurs modest overhead, averaging 2.6% for realistic workloads.

1 Introduction

Ransomware continues to be one of the most important security threats on the Internet. While ransomware is not a new concept (such attacks have been in the wild since the last decade), the growing number of high-profile ransomware attacks [8, 13, 14, 19] has resulted in increasing concerns on how to defend against this class of malware. In 2016, several public and private sectors including the healthcare industry were impacted by ransomware [11, 9, 35]. Recently, US officials have also expressed their concerns about ransomware [16, 20], and even asked the U.S. government to focus on fighting ransomware under the Cybersecurity National Action Plan [20].

In response to the increasing ransomware threat, users are often advised to create backups of their critical data. Certainly, having a reliable data backup policy minimizes the potential costs of being infected with ransomware, and is an important part of the IT management process. However, the growing number of paying victims [10, 29, 17] suggests that unsophisticated users – who are the main target of these attacks – do not follow these recommendations, and easily become a paying victim of ransomware. Hence, ransomware authors continue to create new attacks and evolve their creations as evidenced by the emergence of more sophisticated ransomware every day [34, 7, 33, 32].

Law enforcement agencies and security firms have recently launched a program to assist ransomware victims in retrieving their data without paying ransom fees to cybercriminals [30]. The main idea behind this partnership is that reverse engineers analyze the cryptosystems used by the malware to extract secret keys or find design flaws in the way the sample encrypts or deletes files. While there are ransomware families that are infamous for using weak cryptography [22, 12, 24], newer ransomware variants, unfortunately, have learned from past mistakes by relying on strong cryptographic primitives provided by standard cryptographic libraries. In response to the increasing number of ransomware attacks, a desirable and complementary defense would be to augment the operating system with transparent techniques that would make the operating system resistant against ransomware-like behavior. However, an endpoint approach to defend against unknown ransomware attacks would need to immediately stop attacks once the ransomware starts destroying files, and should be able to recover any lost data.

This paper presents a generic, real-time ransomware protection approach to overcome the limitations of existing approaches with regard to detecting ransomware. Our technique is based on two main components: First, an abstract characterization of the behavior of a large class of current ransomware attacks is constructed. More precisely, our technique applies the results of a long-term dynamic analysis to binary objects to determine if a process matches the abstract model. A process is labeled as *malicious* if it exhibits behaviors that match the abstract model. Second, Redemption employs a high-performance, high-integrity mechanism to protect and restore all attacked files by utilizing a transparent data buffer to redirect access requests while tracking the write contents.

In this paper, we demonstrate that by augmenting the operating system with a set of lightweight and generic techniques, which we collectively call Redemption, it is possible to stop modern ransomware attacks without changing the semantics of the underlying file system's functionality, or performing significant changes in the architecture of the operating system. Our experiments on 29 contemporary ransomware families show that our approach can be successfully applied in an application-transparent manner, and can significantly enhance the current protection capabilities against ransomware (achieving a true positive [TP] rate of 100% at 0.8% false positives [FPs]). Finally, we show that this goal can be achieved without a discernible performance impact, or other changes to the way users interact with standard operating systems. To summarize, we make the following contributions.

- We present a general approach to defending against unknown ransomware attacks in a transparent manner. In this approach, access to user files is mediated, and privileged requests are redirected to a protected area, maintaining the consistent state of user data.
- We show that efficient ransomware protection with zero data loss is possible.
- We present a prototype implementation for Windows, and evaluate it with real users to show that the system is able to protect user files during an unknown ransomware attack while imposing no discernible performance overhead.

The rest of the paper is structured as follows. Section 2 presents related work. In Section 3, we present the threat model. In Section 4, we elaborate on the architecture of Redemption. In Section 6, we provide more details about the implementation of the system. In Section 7, we present the evaluation results. Limitations of the approach are discussed in Section 8. Finally, Section 10 concludes the paper.

2 Related Work

The first scientific study on ransomware was performed by Gazet [18] where he analyzed three ransomware families and concluded that the incorporated techniques in those samples did not fulfill the basic requirements for mass extortion. The recent resurgence of ransomware attacks has attracted the attention of several researchers once more. Kharraz et al. [22] analyzed 15 ransomware families including desktop locker and cryptographic ransomware, and provided an evolution-based study on ransomware attacks. The authors concluded that a significant number of ransomware in the wild has a very similar strategy to attack user files, and can be recognized from benign processes. In another work, Kharraz et al. [21] proposed Unveil, a dynamic analysis system, that is specifically designed to assist reverse engineers to analyze the intrinsic behavior of an arbitrary ransomware sample. Unveil is *not* an end-point solution and no real end-user interaction was involved in their test. Redemption is an end-point solution that aims differentiate between benign and malicious ransomware-like access requests to the file system.

Scaife et al. [31] proposed CryptoDrop which is built upon the premise that the malicious process aggressively encrypts user files. In the paper, as a limitation of CryptoDrop, the authors state that the tool *does not provide any recovery or minimal data loss guarantees*. Their approach is able to detect a ransomware attack after a median of ten file losses. Redemption does not have this limitation as it is designed to protect the consistent state of the original files by providing *full data recovery* if an attack occurs. Hence, unlike CryptoDrop, Redemption guarantees minimal data loss and is resistant to most of realistic evasion techniques that malware authors may use in future.

Very recently, Continella et al. [15], and Kolodenker et al. [23] *concurrently* and *independently* proposed protection schemes to detect ransomware. Continella et al. [15] proposed ShieldFS which has a similar goal to us. The authors also look at the file system layer to find typical ransomware activity. While ShieldFS is a significant improvement over the status quo, it would be desirable to complement it with a more generic approach which is also resistant to unknown cryptographic functions. Unlike ShieldFS, Redemption does not rely on cryptographic primitive identification which can result in false positive cases. More importantly, this was a conscious design choice to minimize the interference with the normal operation of processes, minimize the risk of process crashes and avoid intrusive pop-up prompts which can have noticeable usability side-effects.

Kolodenker et al. [23] proposed PayBreak which securely stores cryptographic encryption keys in a key vault that is used to decrypt affected files after a ransomware attack. In fact, PayBreak intercepts calls to functions that provide cryptographic operations, encrypts symmetric encryption keys, and stores the results in the key vault. After a ransomware attack, the user can decrypt the key vault with his private key and decrypt the files without making any payments. The performance evaluation of the system also shows that PayBreak imposes negligible overhead compared to a reference platform. Similar to ShieldFS, PayBreak relies on identifying functions that implement cryptographic primitives. As mentioned earlier, Redemption does not depend on any hooking technique to identify cryptographic functions. Furthermore, the detection accuracy of Redemption is not impacted by the type of packer a ransomware family may use to evade common anti-malware systems. This makes Redemption a more generic solution to the same problem space.

The evaluation of Redemption covers a significantly larger number of ransomware families compared to [15, 31] and shows it can successfully identify unseen ransomware attacks after observing a median of five exposed files without any data loss. Indeed, Redemption shares some similarity with CryptoDrop, ShieldFS, and PayBreak due to the common characteristics of ransomware attacks. However, extracting such behavior of ransomware is not the main contribution of the paper as they have been comprehensively discussed in several security reports. Rather, Redemption is the introduction of a high performance, data loss free end-user protection framework against ransomware that protects the consistent state of the entire user space and can be used as an augmented service to the operating system. We are not aware of any other scientific work on the protection against ransomware attacks.

3 Threat Model

In this paper, we assume that ransomware can employ any standard, popular techniques to attack machines similar to other types of malware. That is, ransomware can employ several strategies to evade the detection phase, compromise vulnerable machines, and attack the user files. For example, a ransomware instance could be directly started by the user, delivered by a drive-by download attack, or installed via a simple dropper or a malicious email attachment.

We also assume that the malicious process can employ any techniques to generate the encryption key, use arbitrary encryption key lengths, or in general, utilize any customized or standard cryptosystems to lock the files. Ransomware can access sensitive resources by generating new processes, or by injecting code into benign processes (i.e., similarly to other classes of malware). Furthermore, we assume that a user can install and run programs from arbitrary untrusted sources, and therefore, that malicious code can execute with the privileges of the user. This can happen in several scenarios. For instance, a user may install, execute and grant privileges to a malicious application that claims to be a well-known legitimate application, but in fact, delivers malicious payloads – including ransomware.

In addition, in this work, we also assume that the trusted computing base includes the display module, OS kernel, and underlying software and hardware stack. Therefore, we can safely assume that these components of the system are free of malicious code, and that normal user-based access control prevents attackers from running malicious code with superuser privileges. This is a fair assumption considering the fact that ransomware attacks mainly occur in the user-mode.

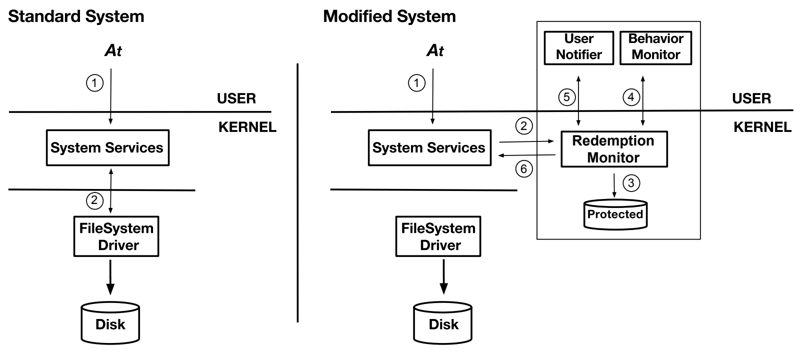

Fig. 1: REDEMPTION mediates the access to the file system and redirects each write request on the user files to a protected area without changing the status of the original file. Reading the user files, creating and writing on *new* files follow the standard 2-step procedure since they do not introduce any risk with regard to ransomware attacks on user data.

4 Design Overview

In this section, we provide our design goals for REDEMPTION. We refer the reader to Section 6 for details of our prototype implementation. REDEMPTION has two main components. First, a lightweight kernel module that intercepts process interactions and stores the event, and manages the changes in a protected area. Second, a user-mode daemon, called behavioral monitor and notification module, that assigns a malice score to a process, and is used to notify the user about the potential malicious behavior of a process.

Intercepting Access Requests. In order to implement a reliable dynamic access control mechanism over user data, this part of the system should be implemented in the kernel, and be able to mediate the access to the file system. The prototype redirects each write access request to the user files to a protected area without changing the status of the original file. We explain more details on how we implemented the write redirection semantics in Section 6.

Figure 1 presents an example that illustrates how access requests are processed. In an unmodified system, the request would succeed if the corresponding file exists, and as long as the process holds the permission. The system introduces the following changes. (1) REDEMPTION receives the request A from the application X to access the file F at the time t, (2) if A_t requests access with write or delete privilege to the file F, and the file F resides in a user defined path, the REDEMPTION's monitor is called, (3) REDEMPTION creates a corresponding file in the protected area, called *reflected* file, and handles the write requests. These changes are periodically flushed to the storage to ensure that they are physically available on the disk. The meta-data entry of the corresponding file is updated with the offset and length of the data buffer in the I/O request after a successful data write at Step 3. (4) the malice score of the process is updated, and is compared to a pre-configured threshold α. (5) the REDEMPTION monitor sends a notification to the display monitor to alert the user depending on the calculated malice score. (6) a success/failure notification is generated, and is sent to the system service manager.

Data Consistency. An important requirement for REDEMPTION is to be able to guarantee data consistency during the interaction of applications with the file system. A natural question that arises here is what happens if the end-user confirms that the suspicious operations on the file that was detected by the system are in fact benign. In this case, having a consistency model is essential to protect the benign changes to the user files without on-disk data corruption. The implementation of the consistency policy should maintain the integrity properties the applications desire from the file system. Failure to do so can lead to corrupted application states and catastrophic data loss. For this reason, the system does not change the file system semantics that may affect the crash guarantees that the file system provides. To this end, REDEMPTION operates in three steps: (1) it reads the meta-data generated for the reflected file, and creates write requests based on the changed data blocks, and changes the status of these blocks to *committed*, (2) upon receiving the confirmation notification, the system updates the meta-data of the reflected file from *committed* to *confirmed*, and (3) the reflected file is deleted from the protected area.

Another question that arises here is how the system protects the consistency of the original file during the above-mentioned three-steps procedure if a system crash occurs. In case of a crash, the system works as follows: (1) if data is committed (Step 1), but the corresponding meta-data is not updated (Step 2), the system treats the change as incomplete, and discards the change as a rollback of an incomplete change. This operation means that Step 2 is partially completed before a crash, so the system repeats the Step 1, (2) If the meta-data of the reflected file is updated to confirmed, it means that the benign changes to the file has been successfully committed to the original file. In this case, the reflected

file is removed from the protected area. Note that a malicious process may attack the Malice Score Calculation (MSC) function by trying to keep the malice score of the process low while performing destructive changes. We elaborate more on these scenarios in Section 8.
User Notification. The trusted output that Redemption utilizes is a visual alert shown whenever a malicious process is detected. We have designed the alert messages to be displayed at the top of the screen to be easily noticeable. Since benign applications usually require sophisticated inputs (i.e., clicking on specific buttons, filling out the path prompt) from the user before performing any sensitive operation on the files, the user is highly likely to be present and interacting with the computer, making it difficult for her to miss an alert.

5 Detection Approach

As mentioned earlier, an important component of Redemption is to perform system-wide application monitoring. For each process that requires privileged access to user files, we assign a *malice score*. The malice score of a process represents the risk that the process exhibits ransomware behavior. That is, the malice score determines whether the Redemption monitor should allow the process to access the files, or notify the user. In the following, we explain the features we used to calculate the malice score of a process. The features mainly target content-based (i.e., changes in the content of each file) and behavior-based (i.e., cross-file behavior of a process) characteristics of ransomware attacks.

5.1 Content-based Features

Entropy Ratio of Data Blocks. For every read and write request to a file, Redemption computes the entropy [25] of the corresponding data buffers in the I/O traces similar to [21]. Comparing the entropy of read and write requests to and from the same file offset serves as an excellent indicator of ransomware behavior. This is due to the popular strategy of reading in the original file data, encrypting it, and writing the encrypted version.
File Content Overwrite. Redemption monitors how a process requests write access to data blocks. In a typical ransomware attack, in order to minimize the chance of recovering files, the malicious process overwrites the content of the user files with random data. Our system increases the malice score of a process as the process requests write access to different parts of a file. In fact, a process is assigned a higher malice score if it overwrites all the content of the files.
Delete Operation. If a process requests to delete a file that belongs to the end-user, it receives a higher malice score. Ransomware samples may not overwrite the data block of the user files directly, but rather generate an encrypted version of the file, and delete the original file.

5.2 Behavior-based Features

Directory Traversal. During an attack, the malicious process often arbitrarily lists user files, and starts encrypting the files with an encryption key. A process

receives a higher malice score if it is iterating over files in a given directory. Note that a typical benign encryption or compression program may also iterate over the files in a directory. However, the generated requests are usually for reading the content of the files, and the encrypted or compressed version of the file is written in a different path. The intuition here is that the ransomware usually intends to lock as many files as possible to force the victim to pay.
Converting to a Specific File Type. A process receives a higher malice score if it converts files of differing types and extensions to a single known or unknown file type. The intuition here is that in many ransomware attacks, unlike most of the benign applications that are specifically designed to operate on specific types of files, the malicious process targets all kinds of user files. To this end, Redemption logs if a process requests access to widely varying classes of files (i.e., videos, images, documents). Note that accessing multiple files with different extensions is not necessarily malicious. Representative examples include the media player to play `.mp3` files (audio) as well as `.avi` (video) files. However, such applications typically open the files with read permission, and more importantly, only generate one request in a short period of time since the application requires specific inputs from the user. Hence, the key insight is that a malicious ransomware process would overwrite or delete the original files.
Access Frequency. If a process frequently generates write requests to user files, we would give this process a higher malice score. We monitor δ – the time between two consequent write access requests on two different user files. Our intuition is that ransomware attacks programmatically list the files and request access to files. Therefore, the δ between two write operations on two different files is not very long – unlike benign applications that usually require some input from the user first in order to perform the required operation.

5.3 Evaluating the Feature Set

Indeed, the assumption that all the features are equally important hardly holds true in real world scenarios. Therefore, we performed a set of measurements to relax this assumption. We used *Recursive Feature Elimination* (RFE) approach to determine the significance of each feature. To this end, the analysis started by incorporating all the features and measuring the FP and TP rates. Then, in each step, a feature with the minimum weight was removed and the FP and TP rates were calculated by performing 10 fold cross-validation to quantify the contribution of each feature. The assigned weights were then used as the coefficient of the feature in the formula 1 in Section 5.4.

Our experiments on several combinations of features shows that the highest false positive rate is 5.9%, and is produced when Redemption only incorporates content-based features (F_1). The reason for this is that file compression applications, when configured to delete the original files, are reported as false positives. During our experiments, we also found out that in document editing programs such as Microsoft Powerpoint or Microsoft Paint, if the user inserts a large image in the editing area, the content-based features that monitor content traversal or payload entropy falsely report the application as being anomalous.

However, when behavior-based features were incorporated, such programs do not receive a high anomaly score since there is no cross-file activities with write privilege similar to ransomware attacks. When all the features are combined (i.e., F_{12}), the minimum false positive rate (0.5% FP with 100% TPs) is produced on labeled dataset. Hence, we use the combination of all the features in our system.

5.4 Malice Score Calculation (MSC) Function

The MSC function allows the system to identify the suspicious process and notify the user when the process matches the abstract model. Given a process X, we assign a malice score S to the process each time it requests privileged access to a user file. If the malice score S exceeds a pre-defined malice threshold α, it means that the process exhibits abnormal behaviors. Hence, we suspend the process and inform the user to confirm the suspicious action. In the following, we provide more details on how we determine the malice score for each process that requests privileged operations on user files:
(r_1): The process that changes the entropy of the data blocks between a read and a write request to a higher value receives a higher malice score. The required value is calculated as an additive inverse of the entropy value of read and write ratio, and resides on [0,1], meaning that the higher the value of entropy in the write operation, the closer the value of the entropy to 1. If the entropy of the data block in write is smaller than the read operation, we assign the value 0 to this feature.
(r_2): If a process iterates over the content of a file with write privilege, it will receive a higher malice score. If the size of the file A is s_A, and y_A is the total size of the data blocks modified by the process, the feature is calculated as $\frac{y_A}{s_A}$ where the higher the number of data blocks modified by the process, the closer the value is to 1.
(r_3): If a process requests to delete a file, this behavior is marked as being suspicious. If a process exhibits such I/O activities, the value 1 is assigned to r_3.
(r_4): Redemption monitors if the process traverses over the user files with write privilege, and computes the additive inverse of the number of privileged accesses to unique files in a given path. The output of the function resides on [0,1]. Given a process X, the function assigns a higher malice score as X generates more write requests to access files in a given path. Here, $write(X, f_i)$ is the i^{th} independent write request generated by the process X on a given file f_i.
(r_5): Given a set of document classes, Redemption monitors whether the process requests *write* access to files that belong to different document classes. The file A and file B belong to two different document classes if the program that opens file A cannot take file B as a valid input. For example, a `docx` and a `pdf` file belong to two different document classes since a `docx` file cannot be opened via a PDF editor program. We assign the score 1 if the process performs cross-document access requests similar to ransomware.
(r_6): The system computes the elapsed time (δ) between two subsequent write requests generated by a single process to access two different files. $\frac{1}{\delta}$ represents

the access frequency. As the elapsed time between two write requests increases, the access frequency decreases.
We define the overall malice score of a process at time t by applying the weights of individual features:

$$MSC(r) = \frac{\sum_{i=1}^{k} w_i \times r_i}{\sum_{i=1}^{k} w_i} \tag{1}$$

where w_i is the predefined weight for the feature i in the MSC function. The value of w_i is based on the experiment discussed in Section 5.3. The weights we used in (1) are $w_1 = 0.9, w_2 = 1.0, w_3 = 0.6, w_4 = 1.0, w_5 = 0.7, w_6 = 1.0$.

Note that when REDEMPTION is active, even when using all the combined features, file encryption or secure deletion applications are typically reported as being suspicious. As mentioned earlier, such applications generate very similar requests to access user files as a ransomware does. For example, in a secure deletion application, the process iterates over the entire content of the given file with write privileges, and writes random payloads on the contents. The same procedure is repeated over the other files in the path. Hence, such cases are reported to the user

as violations, or other inappropriate uses of their critical resources.

6 Implementation

In this section, we provide the implementation details of REDEMPTION. Note that our design is sufficiently general to be applied to any OS that is a potential target for ransomware. However, we built our prototype for the Windows environment which is the main target of current ransomware attacks today.
Monitoring Access Requests. REDEMPTION must interpose on all privileged accesses to sensitive files. The implementation of the system is based on the Windows Kernel Development framework without any modifications on the underlying file system semantics. To this end, it suffices on Windows to monitor the write or delete requests from the I/O system to the base file system driver. Furthermore, to guarantee minimal data loss, REDEMPTION redirects the write requests from the user files to the corresponding reflected files. The reflected files are implemented via *sparse files* on NTFS. In fact, the NTFS file system does not allocate hard disk drive space to reflected files except in regions where they contain non-zero data. When a process requests to open a user file, a sparse file with the same name is created/opened in the protected area. The sparse files are created by calling the function `FltFsControlFile` with the control code `FSCTL_SET_SPARSE`. The size of the file is then set by calling `FltSetInformationFile` that contains the size of the original file.

REDEMPTION updates the `FileName` field in the file object of the create request with the sparse file. By doing this, the system redirects the operation to the reflected file, and the corresponding handle is returned to the requesting process. The write request is executed on the file handle of the reflected file which has been returned to the process at the opening of the file. Each write request contains the offset and the length of the data block that the process wishes to write the data to.

If the write request is successfully performed by the system, the corresponding meta-data of the reflected file (which is the offset and the length of the modified regions of the original file) is marked in the write requests. In our prototype, the meta-data entry to represent the modified regions is implemented via *Reparse Points* provided by Microsoft – which is a collection of application-specific data – and is interpreted by REDEMPTION that sets the tags. When the system sets a reparse point, a unique reparse tag is associated with it which is then used to identify the offset and the length of every change. The reparse point is set by calling `FltTagFile` when the file is created by REDEMPTION. On subsequent accesses to the file in the protected area, the reparse data is parsed via `FltFsControlFile` with the appropriate control code (i.e., `FSCTL_GET_REPARSE_POINT`). Hence, the redirection is achieved by intercepting the original write request, performing the write, and completing the original request while tracking the write contents.

The consistency of the data redirected to the sparse files is an important design requirement of the system. Therefore, it is required to perform frequent flushing to avoid potential user data loss. Indeed, this approach is not without a cost as multiple write requests are required to ensure critical data is written to persistent media. To this end, we use the Microsoft recommended approach by opening sparse files for *unbuffered I/O* upon creation and enabling write-through caching via `FILE_FLAG_NO_BUFFERING` and `FILE_FLAG_WRITE_THROUGH` flags. In fact, with write-through caching enabled, data is still written into the cache, but cache manager writes the data immediately to disk rather than incurring a delay by using the lazy writer. Windows recommends this approach as replacement for calling the `FlushFileBuffer` function after each write which usually causes unnecessary performance penalties in such applications.

Behavioral Detection and Notification Module. We implemented this module as a user-mode service. This was a conscious design choice similar to the design of most anti-malware solutions. Note that Microsoft officially supports the concept of protected services, called Early Launch Anti-Malware (ELAM), to allow anti-malware user-mode services to be launched as protected services. In fact, after the service is launched as a protected service, Windows uses code integrity to only allow trusted code to load into a protected service. Windows also protects these processes from code injection and other attacks from admin processes [28]. If REDEMPTION identifies the existence of a malicious process, it automatically terminates the malicious process.

7 Evaluation

The prototype of the REDEMPTION supports all Windows platforms. In our experiments, we used Windows 7 by simply attaching REDEMPTION to the file system. We took popular anti-evasion measures similar to our experiments in

Chapter 3. The remainder of this section discusses how benign and malicious dataset were collected, and how we conducted the experiments to evaluate the effectiveness of our approach.

7.1 Dataset

The ground truth dataset consists of file system traces of manually confirmed ransomware samples as well as more than 230 GB of data which contains the interaction of benign processes with file system on multiple machines. We used this dataset to verify the effectiveness of Redemption, and to determine the best threshold value to label a suspicious process.
Collecting Ransomware Samples. We collected ransomware samples from public repositories [1, 3] that are updated on a daily basis, and online forums that share malware samples [2, 26]. In total, we collected 9,432 recent samples, and we confirmed 1174 of them to be active ransomware from 29 contemporary ransomware families. We used 504 of the samples from 12 families in our training dataset. Table 2 describes the dataset we used in this experiment.
Collecting Benign Applications. One of the challenges to test Redemption was to collect sufficient amount of benign data, which can represent the realistic use of file system, for model training purposes. To test the proposed approach with realistic workloads, we deployed a version of Redemption on five separate Windows 7 machines in two different time slots each for seven days collecting more that 230 GB of data. The users of the machines were advised to perform their daily activities on their machines. Redemption operated in the monitoring mode, and did not collect any sensitive user information such as credentials, browsing history or personal data. The collected information only included the interaction of processes with the file system which was required to model benign interaction with the file system. All the extracted data was anonymized before performing any further experiments. Based on the collected dataset, we created a pool of application traces that consisted of 65 benign executables including applications that exhibit ransomware-like behavior such as secure deletion, encryption, and compression. The application pool consisted of document editors (e.g., Microsoft Word), audio/video editors (e.g., Microsoft Live Movie Maker, Movavi Video Editor), file compression tools (e.g., Zip, WinRAR), file encryption tools (e.g., AxCrypt, AESCrypt), and popular web browsers (e.g., Firefox, Chrome). Due to space limitation, we provided a sub set of benign applications we used in our analysis in Table 1.

7.2 Detection Results

As discussed in Section 4, one of the design requirements of the system is to produce low false positives, and to minimize the number of unnecessary notifications for the user. To this end, the system employs a threshold value to determine when an end-user should be notified about the suspicious behavior of a process.

We tested a large set of benign as well as ransomware samples on a Redemption enabled machine. As depicted in Table 1 and Table 2, the median score of benign applications is significantly lower than ransomware samples. For file encryption programs such as AxCrypt which are specifically designed to protect the privacy of the users, the original file is overwritten with random data once the encrypted version is generated. In this case, REDEMPTION reports the action as being malicious – which, in fact, is a false positive. Unfortunately, such false positive cases are inevitable since these programs are exhibiting the exact behavior that a typical ransomware exhibits. In such cases, REDEMPTION informs the end-user and asks for a manual confirmation. Given these corner cases, we select the malice score as $\alpha = 0.12$ where the system achieves the best detection and false positive rates (FPs = 0.5% at a TP = 100%). Figure 2 represents the false positive and true positive rates as a function of the malice score on the labeled dataset. This malice threshold is still significantly lower than the minimum malice score of all the ransomware families in the dataset as provided in Table 2. The table also shows the median file recovery rate. As depicted, REDEMPTION detects a malicious process and successfully recovers encrypted data after observing on average four files. Our experiment on the dataset also showed that 7 GB storage is sufficiently large for the protected area in order to enforce the data consistency policy.

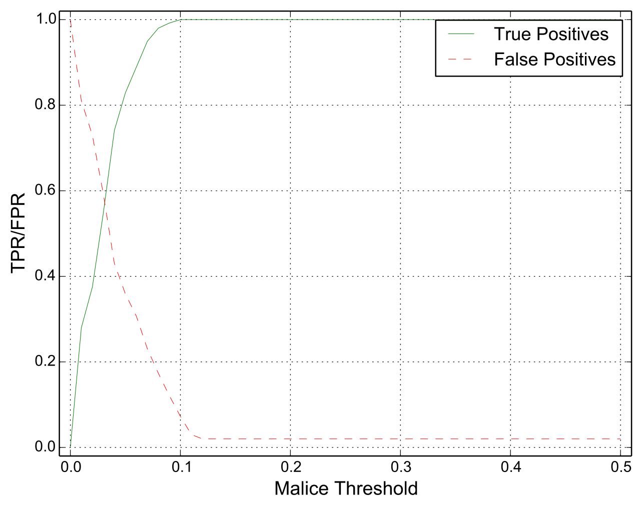

Fig. 2: TP/FP analysis of REDEMPTION. The threshold value $\alpha = 0.12$ gives the best detection and false positive rates (FPs = 0.5% at a TP = 100%).

Testing with Known/Unknown Samples. In addition to the 10-fold cross validation on 504 samples, we also tested REDEMPTION with unknown benign and malicious dataset. *The tests included 29 ransomware families which 57% of them were not presented in the training dataset.* We also incorporated the file

Table 1: A list of Benign application and their malice scores.

Program	Min. Score	Max. Score
Adobe Photoshop	0.032	0.088
AESCrypt	0.37	0.72
AxCrypt	0.31	0.75
Adobe PDF reader	0.0	0.0
Adobe PDF Pro	0.031	0.039
Google Chrome	0.037	0.044
Internet Explorer	0.035	0.045
Matlab	0.038	0.92
MS Words	0.041	0.089
MS PowerPoint	0.025	0.102
MS Excel	0.017	0.019
VLC Player	0.0	0.0
Vera Crypt	0.33	0.71
WinRAR	0.0	0.16
Windows Backup	0.0	0.0
Windows paintit	0.029	0.083
SDelete	0.283	0.638
Skype	0.011	0.013
Spotify	0.01	0.011
Sumatra PDF	0.022	0.041
Zip	0.0	0.16
Malice Score Median	0.027	0.0885

Table 2: A list of ransomware families and their malice scores.

Family	Samples	Min. Score	Max. Score	File Recovery
Cerber	33	0.41	0.73	5
Cryptolocker	50	0.36	0.77	4
CryptoWall3	39	0.4	0.79	6
CryptXXX	46	0.49	0.71	3
CTB-Locker	53	0.38	0.75	7
CrypVault	36	0.53	0.73	3
CoinVault	39	0.42	0.69	4
Filecoder	54	0.52	0.66	5
GpCode	45	0.52	0.76	2
TeslaCrypt	37	0.43	0.79	4
Virlock	29	0.51	0.72	3
SilentCrypt	43	0.31	0.59	9
Total Samples	**504**	-	-	-
Score Median	-	**0.43**	**0.73**	-
File Recovery Median	-	-	-	4

system traces of benign processes in the second time slot as discussed in Section 7.1 as the unseen benign dataset in this test. Table 3 represents the list of ransomware families we used in our experiments. This table also shows the datasets that were used in prior work [15, 31, 23]. In this experiment, we used the malice threshold $\alpha = 0.12$ similar to the previous experiment and manually checked the detection results to measure the FP and TP rates. The detection results in this set of experiments is (TPs = 100% at 0.8% FPs). Note that the number of FP cases depends on the value of malice threshold. We selected this conservative value to be able to detect all the possible ransomware behaviors. Indeed, observing realistic work loads on a larger group of machines can lead to a more comprehensive model, more accurate malice threshold calibration, and ultimately lower FP rates. However, our experiments on 677 ransomware samples from 29 ransomware families show that REDEMPTION is able to detect the malicious process in *all* the 29 families by observing a median of 5 files. We suspect the difference in the number of files is due to difference in the size of the files being attacked. In fact, this is a very promising result since the detection rate of the system did not change by adding unknown ransomware families which do not necessarily follow the same attack techniques (i.e., using different cryptosystems). *The results of this experiment also shows that the number of exposed files to ransomware does not change significantly if* REDEMPTION *is not trained with unseen ransomware families.* This result clearly implies that the system can detect a significant number of *unseen* ransomware attacks.

7.3 Disk I/O and File System Benchmarks

In order to evaluate the disk I/O and file system performance of REDEMPTION, we used IOzone [6], a well-known file system benchmark tool for Windows. To

Family	**Redemption Samples/FA**	**CryptoDrop [31] Samples/FA**	**ShieldFS [15] Samples**	**PayBreak [23] Samples**
Almalocker	-	-	-	1
Androm	-	-	-	2
Cerber	30/6	-	-	1
Chimera	-	-	-	1
CoinVault	19/5	-	-	-
Critroni	16/6	-	17	-
Crowti	22/8	-	-	-
CryptoDefense	42/7	18/6.5	6	-
CryptoLocker(copycat)	-	2/20	-	-
Cryptolocker	29/4	31/10	20	33
CryptoFortess	12/7	2/14	-	2
CryptoWall	29/5	8/10	8	7
CrypWall	-	-	-	4
CrypVault	26/3	-	-	-
CryptXXX	45/3	-	-	-
CryptMIC	7/3	-	-	-
CTB-Locker	33/6	122/29	-	-
DirtyDecrypt	8/3	-	3	-
DXXD	-	-	-	2
Filecoder	34/5	72/10	-	-
GpCode	45/3	13/22	-	2
HDDCryptor	13/5	-	-	-
Jigsaw	12/4	-	-	-
Locky	21/2	-	154	7
MarsJokes	-	-	-	1
MBL Advisory	12/4	1/9	-	-
Petya	32/5	-	-	-
PayCrypt	-	-	3	-
PokemonGo	-	-	-	1
PoshCoder	17/4	1/10	-	-
TeslaCrypt	39/6	149/10	73	4
Thor Locky	-	-	-	1
TorrentLocker	21/6	1/3	12	-
Tox	15/7	-	-	9
Troldesh	-	-	-	5
Virlock	29/7	20/8	-	4
Razy	-	-	-	3
SamSam	-	-	-	4
SilentCrypt	43/8	-	-	-
Xorist	14/7	51/3	-	-
Ransom-FUE	-	1/19	-	-
WannaCry	7/5	-	-	-
ZeroLocker	5/8	-	1	-
Total Samples (Families)	**677(29)**	**492(15)**	**305(11)**	**107(20)**
File Attacked/Recovered(FA/FR) Median	**5/5**	**10/0**	-	-

Table 3: The list of ransomware families used to test REDEMPTION, CryptoDrop [31], ShieldFS [15], and PayBreak [23]. The numbers shown for [31, 15, 23] are extracted from the corresponding papers.

this end, we first generated 100 × 512 MB files to test the throughput of block write, rewrite, and read operations. Next, we tested the standard file system operations by creating and accessing 50,200 files, each containing 1 MB of data in multiple directories. We ran IOzone as a normal process. Then, for having a comparison, we repeated all the experiments 10 times, and calculated the average scores to get the final results. We wrote a script in AutoIt [5] to automate the tasks.The results of our findings are summarized in Table 4.

The experiments show that REDEMPTION performs well when issuing heavy reads and writes, and imposes an overhead of 2.8% and 3.4%, respectively. However, rewrite and create operations can experience slowdowns ranging from 7% to 9% when dealing with a large number of small files. In fact, creating the reflected files and redirecting the write requests to the protected area are the main reasons of this performance hit under high workloads. These results also suggest that REDEMPTION might not be suitable for workloads involving many small files such as compiling large software projects. However, note that such heavy workloads do not represent the deployment cases Redemption is designed to target (i.e., protecting the end host of a typical user that surfs the web and engages in productivity activities such as writing text and sending emails).

Another important question that arises here is that how many files should be maintained in the protected area when Redemption is active. In fact, as the protected area is sufficiently large, the system can maintain several files without committing them to the disk and updating the original files. However, this approach may not be desirable in scenarios where several read operations may occur immediately after write operations (i.e., database). More specifically, in these scenarios, Redemption, in addition to write requests, Redemption should also redirect read operations to the protected area which is not ideal from usability perspective. To this end, we also performed an I/O benchmarking on the protected area by requesting write access to files, updating the files, and committing the changes to the protected area without updating the original files. We created a script to immediately generate read requests to access updated files. The I/O benchmark on the protected area shows that the performance overhead for read operations is less than 3.1% when 100 files with median file size of 17.4 MB are maintained in the protected area. This number of files is significantly larger than the maximum number of files Redemption needs to observe to identify the suspicious process. Note that we consider the scenarios where read operations are requested immediately after write operations to exercise the redirection mechanism under high loads. Based on this performance benchmarking, we conclude that read redirection mechanism does not impose a significant overhead as we first expected. In the following, we demonstrate that Redemption incurs minimal performance overhead when executing more realistic workloads for our target audience.

7.4 Real-world Application Testing

To obtain measurable performance indicators to characterize the overhead of Redemption, we created micro-benchmarks that exercise the critical performance paths of Redemption. Note that developing benchmarks and custom test cases requires careful consideration of factors that might impact the runtime measurements. For example, a major challenge we had to tackle was automating the testing of desktop applications with graphical user interfaces. In order to perform the tests as identical as possible on the standard and Redemption-enabled machines, we wrote scripts in AutoIt to interact with each application while monitoring their performance impact. To this end, we called the application within the script, and waited for 5 seconds for the program window to appear. We then automatically checked whether the GUI of the application is the active window. The script forced the control's window of the application to be on top. We then started interacting with the edit control and other parts of the programs to exercise the core features of the applications using the handle returned by the AutoIt script. Similarly to the previous experiment, we repeated each test 10 times. We present the average runtimes in Table 5.

Table 4: Disk I/O performance in a standard and a Redemption-protected host.

Operation	Original Performance	Redemption Performance	Overhead(%)
Write	112,456.25 KB/s	110094.67KB/s	3.4%
Rewrite	68,457.57 KB/s	62501.76 KB/s	8.7%
Read	114,124.78 KB/s	112070.53 KB/s	2.8%
Create	12,785 files/s	11,852 files/s	7.3%

Table 5: Runtime overhead of Redemption on a set of end-point applications

Application	Original (s)	Redemption (s)	Overhead (%)
AESCrypt	165.55	173.28	4.67%
AxCrypt	182.4	191.72	5.11%
Chrome	66.19	67.02	1.25%
IE	68.58	69.73	1.67%
Media Player	118.2	118.78	0.49%
MS Paint	134.5	138.91	3.28%
MS Word	182.17	187.84	3.11%
SDelete	219.4	231.0	5.29%
Vera Crypt	187.5	196.46	4.78%
Winzip	139.7	141.39	1.21%
WinRAR	160.8	163.12	1.44%
zip	127.8	129.32	1.19%
Average	-	-	2.6%

In our experiments, the overhead of protecting a system from ransomware was under 6% in every test case, and, on average, running applications took only 2.6% longer to complete their tasks. These results demonstrate that Redemption is efficient, and that it should not detract from the user experience. These experiments also support that Redemption can provide real time protection against ransomware without a significant performance impact. We must stress that if Redemption is deployed on machines with a primarily I/O bound workload, lower performance should be expected as indicated by the benchmark in Section 7.3.

7.5 Usability Experiments

We performed a user study experiment with 28 participants to test the usability of Redemption. We submitted and received IRB waiver for our usability experiments from the office of Human Subject Research Protection (HSRP). The goal of the usability test is to determine whether the system provides transparent monitoring, and also to evaluate how end-users deal with our visual alerts. The participants were from different majors at the authors' institution. Participants were recruited by asking for volunteers to help test a security tool. In order to avoid the effects of priming, the participants were not informed about the key functionality of Redemption. The recruitment requirement was that the participants are familiar with text editors and web browsers so that they could perform the given tasks correctly. All the experiments were conducted using two identical Windows 7 virtual machines enabled with Redemption on two laptops. The virtual machines were provided a controlled Internet access as described in Section 7. Redemption was configured to be in the protection mode on the entire data space generated for the test user account. A ransomware sample was automatically started at a random time to observe how the user interacts with Redemption during a ransomware attack. After each experiment, the virtual machines were rolled back to the default state. No personal information was collected from the participants at *any* point of the experiments.

We asked the participants to perform three tasks to evaluate different aspects of the system. The first task was to work with an instance of Microsoft Word and PowerPoint on the test machines running REDEMPTION. The experiment observer asked the participants to compare this process with their previous experience of using Microsoft Word and PowerPoint and rate the difficulty involved in interacting with the test setup on a 5-point Likert scale.

In the second task, the participants were asked to encrypt a folder containing multiple files with AxCrypt on the REDEMPTION-enabled machine. This action caused a visual alert to be displayed to the participant that the operation is suspended, and ask the user to confirm or deny the action. The participants were asked to explain why they confirmed or denied the action and the reason behind their decision.

In the last task, the participants were asked to perform a specific search on the Internet. While they were pre-occupied with the task, the ransomware sample was automatically started. This action was blocked by REDEMPTION and caused another visual alert to be displayed. Similar to the second task, the experiment observer monitored how participants handled the alert.

At the end of the first phase of the experiment, all 28 participants found the experience to be identical to using Microsoft Word and PowerPoint on their own machines. This finding empirically confirms that REDEMPTION is transparent to the users. In the second experiment, 26 participants confirmed the action. Another 2 noticed the alert, but denied the operation so no file was encrypted. In the third phase, all the 28 participants noticed the visual alert, and none of the users confirmed the operation. The participants explained that they were not sure why they received this visual alert, and could not verify the operation. These results confirm that REDEMPTION visual alerts are able to draw all participants' attention while they are occupied with other tasks, and are effective in protecting the user data. Furthermore, the experiments clearly imply that end-users are more likely to recognize the presence of suspicious operations on their sensitive data using REDEMPTION indicators. To confirm statistical significance, we performed a hypothesis test where the null hypothesis is that REDEMPTION's indicators do not assist in identifying suspicious operations during ransomware attacks, while the alternative hypothesis is that REDEMPTION's ransomware indicators do assist in identifying such destructive actions. Using a paired t-test, we obtain a p-value of 4.9491×10^{-7}, sufficient to reject the null hypothesis at a 1% significance level.

8 Discussion and Limitations

Unfortunately, malware research is an arms race. Therefore, there is always the possibility that malware developers find heuristics to bypass the detection on the analysis systems, or on end-user machines. In the following, we discuss possible evasion scenarios that can be used by malware authors, and how REDEMPTION addresses them.

Attacking Redemption's Monitor.

Note that the interaction of any user-mode process as well as kernel mode drivers with the file system is managed by Windows I/O manager which is responsible for generating appropriate I/O requests. Since every access in any form should be first submitted to the I/O manager, and REDEMPTION registers call-backs to all the I/O requests, bypassing REDEMPTION's monitor is not possible in the user-mode. Furthermore, note that direct access to the disk or volume is prohibited by Windows from Windows Vista [27] for user-mode applications in order to protect file system's integrity. Therefore, any other form of requests to access the files is not possible in the user-mode, and is guaranteed by the operating system.

Attackers may be able to use social engineering techniques and frustrate users by creating fake alert messages – accusing a browser to be a ransomware – and forcing the user to turn off REDEMPTION. We believe these scenarios are possible. However, note that such social engineering attacks are well-known security problems and target *all* end-point security solutions including our tool. Defending against such attacks depends more on the security awareness of users and is out of scope of this work.

Attacking the Malice Score Calculation Function.

An attacker may also target the malice calculation function, and try to keep the malice score of the process lower than the threshold. For example, an attacker can generate code that performs *selective* content overwrite, use a low entropy payload for content overwrite, or launch periodic file destruction. If an attacker employs any one of these techniques *by itself*, the malice score becomes lower, but the malicious action would still be distinguishable. For example, if the file content is overwritten with low entropy payload, the process receives a lower malice score. However, since the process overwrites all the content of a file with a low-entropy payload, it is itself suspicious, and would be reported to the user.

We believe that the worst case scenario would be if an attacker employs all the three techniques simultaneously to bypass the malice score calculation function. This is a fair assumption since developing such a malware is straightforward. However, note that in order to launch a successful ransomware attack, and force the victim to pay the ransom fee, the malicious program needs to attack more than a file – preferably all the files on the system. Hence, even if the malicious program employs all of the bypassing techniques, it requires some sort of *iteration* with write permission over the user files. This action would still be seen and captured by REDEMPTION. In this particular case, a malicious program can successfully encrypt a *single* user file, but the subsequent write attempt on another file would be reported to the user for the confirmation if the write request occurs within a pre-defined six hour period after the first attempt. This means a ransomware can successfully encrypt a user file every six hours. We should stress that, in this particular scenario, the system cannot guarantee zero data loss. However, the system significantly decreases the effectiveness of the attack since the number of files encrypted per day is very small.

Furthermore, since these approaches incur a considerable delay to launch a successful attack, they also increase the risk of being detected by AV scanners on the end-point before encrypting a large number of files, and forcing the user to pay. Consequently, developing such *stealthy* ransomware may not be as profitable as current ransomware attack strategies where the entire point of the attack is to encrypt as many files as possible in a short period of time and request money. An attacker may also avoid performing user file encryption, and only lock the desktop once installed. This approach can make the end-user machine inaccessible. However, such changes are not persistent, and regaining access to the machine is significantly easier, and is out of the scope of this paper.

9 Acknowledgements

This work was supported by the National Science Foundation (NSF) under grant CNS-1409738, and Secure Business Austria.

10 Conclusions

In this paper, we proposed a generic approach, called REDEMPTION, to defend against ransomware on the end-host. We show that by incorporating the prototype of REDEMPTION as an augmented service to the operating system, it is possible to successfully stop ransomware attacks on end-user machines. We showed that the system incurs modest overhead, averaging 2.6% for realistic workloads. Furthermore, REDEMPTION does not require explicit application support or any other preconditions to actively protect users against unknown ransomware attacks. We provide an anonymous video of REDEMPTION in action in [4], and hope that the concepts we propose will be useful for end-point protection providers.

References

1. Minotaur Analysis - Malware Repository. minotauranalysis.com.
2. Malware Tips - Your Security Advisor. http://malwaretips.com/forums/virus-exchange.104/.
3. MalwareBlackList - Online Repository of Malicious URLs. http://www.malwareblacklist.com.
4. A brief demo on how Redemption operates. https://www.youtube.com/watch?v=iuEgFVz7a7g, 2016.
5. AutoIt. https://www.autoitscript.com/site/autoit/, 2016.
6. IOzone Filesystem Benchmark. www.iozone.org, 2016.
7. AJJAN, A. Ransomware: Next-Generation Fake Antivirus. http://www.sophos.com/en-us/medialibrary/PDFs/technicalpapers/SophosRansomwareFakeAntivirus.pdf, 2013.
8. ALEX HERN. Major sites including New York Times and BBC hit by ransomware malvertising. https://www.theguardian.com/technology/2016/mar/16/major-sites-new-york-times-bbc-ransomware-malvertising, 2016.
9. ALEX HERN. Ransomware threat on the rise as almost 40 percent of bussinesses attacked. https://www.theguardian.com/technology/2016/aug/03/ransomware-threat-on-the-rise-as-40-of-businesses-attacked, 2016.
10. ANDREW DALTON. Hospital paid 17K ransom to hackers of its computer network. http://bigstory.ap.org/article/d89e63ffea8b46d98583bfe06cf2c5af/hospital-paid-17k-ransom-hackers-its-computer-network, 2016.

11. BBC News. University pays 20,000 Dollars to ransomware hackers. `http://www.bbc.com/news/technology-36478650`, 2016.
12. Charlie Osborne. Researchers launch another salvo at CryptXXX ransomware. `http://www.zdnet.com/article/researchers-launch-another-salvo-at-cryptxxx-ransomware/`, 2016.
13. Chris Francescani. Ransomware Hackers Blackmail U.S. Police Departments. `http://www.cnbc.com/2016/04/26/ransomware-hackers-blackmail-us-police-departments.html`, 2016.
14. Connor Mannion. Three U.S. Hospitals Hit in String of Ransomware Attacks. `http://www.nbcnews.com/tech/security/three-u-s-hospitals-hit-string-ransomware-attacks-n544366`, 2016.
15. Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., and Maggi, F. Shieldfs: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (2016), ACM, pp. 336–347.
16. Dan Whitcomb. California lawmakers take step toward outlawing ransomware. `http://www.reuters.com/article/us-california-ransomware-idUSKCN0X92PA`, 2016.
17. Dell SecureWorks. University of Calgary paid 20K in ransomware attack. `http://www.cbc.ca/news/canada/calgary/university-calgary-ransomware-cyberattack-1.3620979`, 2016.
18. Gazet, A. Comparative analysis of various ransomware virii. *Journal in Computer Virology 6* (2010), 77–90.
19. Grefgory Wolf. 8 High Profile Ransomware Attacks You May Not Have Heard Of. `https://www.linkedin.com/pulse/8-high-profile-ransomware-attacks-you-may-have-heard-gregory-wolf`, 2016.
20. Jerry Zremski. New York Senator Seeks to Combat Ransomware. `http://www.govtech.com/security/New-York-Senator-Seeks-to-Combat-Ransomware.html`, 2016.
21. Kharraz, A., Arshad, S., Mulliner, C., Robertson, W., and Kirda, E. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In *25th USENIX Security Symposium* (2016).
22. Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L., and Kirda, E. Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)* (2015).
23. Kolodenker, E., Koch, W., Stringhini, G., and Egele, M. Paybreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (New York, NY, USA, 2017), ASIA CCS '17, ACM, pp. 599–611.
24. Lawrance Abrams. TeslaCrypt Decrypted: Flaw in TeslaCrypt allows Victim's to Recover their Files. `http://www.bleepingcomputer.com/news/security/teslacrypt-decrypted-flaw-in-teslacrypt-allows-victims-to-recover-their-files/`, 2016.
25. Lin, J. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory 37* (1991), 145–151.
26. Malware Don't Need Coffee. Guess who's back again ? Cryptowall 3.0. `http://malware.dontneedcoffee.com/2015/01/guess-whos-back-again-cryptowall-30.html`, 2015.

27. Microsoft, Inc. Blocking Direct Write Operations to Volumes and Disks. `https://msdn.microsoft.com/en-us/library/windows/hardware/ff551353(v=vs.85).aspx`.
28. Microsoft, Inc. Protecting Anti-Malware Services. `https://msdn.microsoft.com/en-us/library/windows/desktop/dn313124(v=vs.85).aspx`, 2016.
29. Ms. Smith. Kansas Heart Hospital hit with ransomware; attackers demand two ransoms. `http://www.networkworld.com/article/3073495/security/kansas-heart-hospital-hit-with-ransomware-paid-but-attackers-demanded-2nd-ransom.html`, 2016.
30. No-More-Ransomware Project. No More Ransomware! `https://www.nomoreransom.org/about-the-project.html`, 2016.
31. Nolen Scaife, Henry Carter, P. T., and Butler, K. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In *In IEEE International Conference on Distributed Computing Systems (ICDCS)* (2016).
32. O'Gorman, G., and McDonald, G. Ransomware: A Growing Menance. `http://www.symantec.com/connect/blogs/ransomware-growing-menace`, 2012.
33. Symantec, Inc. Internet Security Threat Report. `http://www.symantec.com/security_response/publications/threatreport.jsp`, 2014.
34. TrendLabs. An Onslaught of Online Banking Malware and Ransomware. `http://apac.trendmicro.com/cloud-content/apac/pdfs/security-intelligence/reports/rpt-cashing-in-on-digital-information.pdf`, 2013.
35. WIRED Magazine. Why Hospitals Are the Perfect Targets for Ransomware. `https://www.wired.com/2016/03/ransomware-why-hospitals-are-the-perfect-targets/`, 2016.