UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware

Amin Kharraz
Northeastern University

Sajjad Arshad
Northeastern University

Collin Mulliner
Northeastern University

William Robertson
Northeastern University

Engin Kirda
Northeastern University

Abstract

Although the concept of ransomware is not new (i.e., such attacks date back at least as far as the 1980s), this type of malware has recently experienced a resurgence in popularity. In fact, in the last few years, a number of high-profile ransomware attacks were reported, such as the large-scale attack against Sony that prompted the company to delay the release of the film "The Interview." Ransomware typically operates by locking the desktop of the victim to render the system inaccessible to the user, or by encrypting, overwriting, or deleting the user's files. However, while many generic malware detection systems have been proposed, none of these systems have attempted to specifically address the ransomware detection problem.

In this paper, we present a novel dynamic analysis system called UNVEIL that is specifically designed to detect ransomware. The key insight of the analysis is that in order to mount a successful attack, ransomware must tamper with a user's files or desktop. UNVEIL automatically generates an artificial user environment, and detects when ransomware interacts with user data. In parallel, the approach tracks changes to the system's desktop that indicate ransomware-like behavior. Our evaluation shows that UNVEIL significantly improves the state of the art, and is able to identify previously unknown evasive ransomware that was not detected by the anti-malware industry.

1 Introduction

Malware continues to remain one of the most important security threats on the Internet today. Recently, a specific form of malware called ransomware has become very popular with cybercriminals. Although the concept of ransomware is not new – such attacks were registered as far back as the end of the 1980s – the recent success of ransomware has resulted in an increasing number of new families in the last few years [7, 20, 21, 44, 46]. For example, CryptoWall 3.0 made headlines around the world as a highly profitable ransomware family, causing an estimated $325M in damages [45]. As another example, the Sony ransomware attack [27] received large media attention, and the U.S. government even took the official position that North Korea was behind the attack.

Ransomware operates in many different ways, from simply locking the desktop of the infected computer to encrypting all of its files. Compared to traditional malware, ransomware exhibits behavioral differences. For example, traditional malware typically aims to achieve stealth so it can collect banking credentials or keystrokes without raising suspicion. In contrast, ransomware behavior is in direct opposition to stealth, since the entire point of the attack is to openly notify the user that she is infected.

Today, an important enabler for behavior-based malware detection is dynamic analysis. These systems execute a captured malware sample in a controlled environment, and record its behavior (e.g., system calls, API calls, and network traffic). Unfortunately, malware detection systems that focus on stealthy malware behavior (e.g., suspicious operating system functionality for keylogging) might fail to detect ransomware because this class of malicious code engages in activity that appears similar to benign applications that use encryption or compression. Furthermore, these systems are currently not well-suited for detecting the specific behaviors that ransomware engages in, as evidenced by misclassifications of ransomware families by AV scanners [10, 39].

In this paper, we present a novel dynamic analysis system that is designed to analyze and detect ransomware attacks and model their behaviors. In our approach, the system automatically creates an artificial, realistic execution environment and monitors how ransomware interacts with that environment. Closely monitoring process interactions with the filesystem allows the system to precisely characterize cryptographic ransomware behavior.

In parallel, the system tracks changes to the computer's desktop that indicates ransomware-like behavior. The key insight is that in order to be successful, ransomware will need to access and tamper with a victim's files or desktop. Our automated approach, called UNVEIL, allows the system to analyze many malware samples at a large scale, and to reliably detect and flag those that exhibit ransomware-like behavior. In addition, the system is able to provide insights into how the ransomware operates, and how to automatically differentiate between different classes of ransomware.

We implemented a prototype of UNVEIL in Windows on top of the popular open source malware analysis framework Cuckoo Sandbox [13]. Our system is implemented through custom Windows kernel drivers that provide monitoring capabilities for the filesystem. Furthermore, we added components that run outside the sandbox to monitor the user interface of the target computer system.

We performed a long-term study analyzing 148,223 recent general malware samples in the wild. Our large-scale experiments show that UNVEIL was able to correctly detect 13,637 ransomware samples from multiple families in live, real-world data feeds with no false positives. Our evaluation also suggests that current malware analysis systems may not yet have accurate behavioral models to detect different classes of ransomware attacks. For example, the system was able to correctly detect 7,572 ransomware samples that were previously unknown and undetected by traditional AVs, but belonged to modern file locker ransomware families. UNVEIL was also able to detect a new type of ransomware that had not previously been reported by any security company. This ransomware also did not show any malicious activity in a modern sandboxing technology provided by a well-known anti-malware company, while showing heavy file encryption activity when analyzed by UNVEIL.

The high detection rate of our approach suggests that UNVEIL can complement current malware analysis systems to quickly identify new ransomware samples in the wild. UNVEIL can be easily deployed on any malware analysis system by simply attaching to the filesystem driver in the analysis environment.

In summary, this paper makes the following contributions:

- We present a novel technique to detect ransomware known as *file lockers* that targets files stored on a victim's computer. Our technique is based on monitoring system-wide filesystem accesses in combination with the deployment of automatically-generated artificial user environments for triggering ransomware.
- We present a novel technique to detect ransomware known as *screen lockers*. Such ransomware prevents access to the computer system itself. Our technique is based on detecting locked desktops using dissimilarity scores of screenshots taken from the analysis system's desktop before, during, and after executing the malware sample.
- We performed a large-scale evaluation to show that our approach can effectively detect ransomware. We automatically detected and verified 13,637 ransomware samples from a dataset of 148,223 recent general malware. In addition, we found one previously unknown ransomware sample that does not belong to any previously reported family. Our evaluation demonstrates that our technique works well in practice (achieving a true positive [TP] rate 96.3% at zero false positives [FPs]), and is useful in automatically identifying ransomware samples submitted to analysis and detection systems.

The rest of the paper is structured as follows. In Section 2, we briefly present background information and explain different classes of ransomware attacks. In Section 3, we describe the architecture of UNVEIL and explain our detection approaches for multiple types of ransomware attacks. In Section 4, we provide more details about our dynamic analysis environment. In Section 5, we present the evaluation results. Limitations of the approach are discussed in Section 6, while Section 7 presents related work. Finally, Section 8 concludes the paper.

2 Background

Ransomware, like other classes of malware, uses a number of strategies to evade detection, propagate, and attack users. For example, it can perform multi-infection or process injection, exfiltrate the user's information to a third party, encrypt files, and establish secure communication with C&C servers. Our detection approach assumes that ransomware samples can and will use all of the techniques that other malware samples may use. In addition, our system assumes that successful ransomware attacks perform one or more of the following activities.

Persistent desktop message. After successfully performing a ransomware infection, the malicious program typically displays a message to the victim. This "ransom note" informs the users that their computer has been "locked" and provides instructions on how to make a ransom payment to restore access. This ransom message can be generated in different ways. A popular technique is to call dedicated API functions (e.g., `CreateDesktop()`) to create a new desktop and make it the default config-

uration to lock the victim out of the compromised system. Malware writers can also use HTML or create other forms of persistent windows to display this message. Displaying a persistent desktop message is a classic action in many ransomware attacks.

Indiscriminate encryption and deletion of the user's private files. A crypto-style ransomware attack lists the victim's files and aggressively encrypts any private files it discovers. Access is restricted by withholding the decryption key. Encryption keys can be generated locally by the malware on the victim's computer, or remotely on C&C servers, and then delivered to the compromised computer. An attacker can use customized destructive functions, or Windows API functions to delete the original user's files. The attacker can also overwrite files with the encrypted version, or use secure deletion via the Windows Secure Deletion API.

Selective encryption and deletion of the user's private files based on certain attributes (e.g., size, date accessed, extension). In order to avoid detection, a significant number of ransomware samples encrypt a user's private files selectively. In the simplest form, the ransomware sample can list the files based on the access date. In more sophisticated scenarios, the malware could also open an application (e.g., `word.exe`) and list recently accessed files. The sample can also inject malicious code into any Windows application to obtain this type of information (e.g., directly reading process memory).

In this work, we address all of these scenarios where an adversary has already compromised a system, and is able to launch arbitrary ransomware-related operations on the user's files or desktop.

3 UNVEIL Design

In this section, we describe our techniques for detecting multiple classes of ransomware attacks. We refer the reader to Section 4 for details on the implementation details of the prototype.

3.1 Detecting File Lockers

We first describe why our system creates a unique, artificial user environment in each malware run. We then present the design of the filesystem activity monitor and describe how UNVEIL uses the output of the filesystem monitor to detect ransomware.

3.1.1 Generating Artificial User Environments

Protecting malware analysis environments against fingerprinting techniques is non-trivial in a real-world deployment. Sophisticated malware authors exploit static features inside analysis systems (e.g., name of a computer) and launch reconnaissance-based attacks [31] to fingerprint both public and private malware analysis systems.

The static features of analysis environments can be viewed as the Achilles' heel of malware analysis systems. One static feature that can have a significant impact on the effectiveness of the malware analysis systems is the user data that can be effectively used to fingerprint the analysis environment. That is, even on bare-metal environments where classic tricks such as virtualization checks are not possible, an unrealistic looking user environment can be a telltale sign that the code is running in a malware analysis system.

Intuitively, a possible approach to address such reconnaissance attacks is to build the user environment in such a way that the user data is valid, real, and non-deterministic in each malware run. These automatically-generated user environments serve as an "enticing target" to encourage ransomware to attack the user's data while at the same time preventing the possibility of being recognized by adversaries.

In practice, generating a user environment is a non-trivial problem, especially if this is to be done automatically. This is because the content generator should not allow the malware author to fingerprint the automatically-generated user content located in the analysis environment, and also determine that it does not belong to a real user. We elaborate on how we automatically generate an artificial – yet realistic – user environment for ransomware in each malware run in Section 4.1.

3.1.2 Filesystem Activity Monitor

The filesystem monitor in UNVEIL has direct access to data buffers involved in I/O requests, giving the system full visibility into all filesystem modifications. Each I/O operation contains the process name, timestamp, operation type, filesystem path and the pointers to the data buffers with the corresponding entropy information in read/write requests. The generation of I/O requests happens at the lowest possible layer to the filesystem. For example, there are multiple ways to read, write, or list files in user-/kernel-mode, but all of these functions are ultimately converted to a sequence of I/O requests. Whenever a user thread invokes an I/O API, an I/O request is generated and is passed to the filesystem driver. Figure 1 shows a high-level design of UNVEIL in the Windows environment.

UNVEIL's monitor sets callbacks on all I/O requests to the filesystem generated on behalf of any user-mode processes. We note that for UNVEIL operations, it is desirable to only set one callback per I/O request for performance reasons, and that this also maintains full visibility into I/O operations. In UNVEIL, user-mode process in-

teractions with the filesystem are formalized as access patterns. We consider access patterns in terms of I/O traces, where a trace T is a sequence of t_i such that

$t_i = \langle P, F, O, E \rangle$,

P is the set of user-mode processes,

F is the set of available files,

O is the set of I/O operations, and

E is the entropy of read or write data buffers.

For all of the file locker ransomware samples that we studied, we empirically observed that these samples issue I/O traces that exhibit distinctive, repetitive patterns. This is due to the fact that these samples each use a single, specific strategy to deny access to the user's files. This attack strategy is accurately reflected in the form of I/O access patterns that are repeated for each file when performing the attack. Consequently, these I/O access patterns can be extracted as a distinctive I/O fingerprint for a particular family. We note that our approach mainly considers write or delete requests. We elaborate on extracting I/O access patterns per file in Section 3.1.2.

I/O Data Buffer Entropy. For every read and write request to a file captured in an I/O trace, UNVEIL computes the entropy of the corresponding data buffer. Comparing the entropy of read and write requests to and from the same file offset serves as an excellent indicator of crypto-ransomware behavior. This is due to the common strategy to read in the original file data, encrypt it, and overwrite the original data with the encrypted version. The system uses Shannon entropy [30] for this computation. In particular, assuming a uniform random distribution of bytes in a data block d, we have

$$H(d) = -\sum_{i=1}^{n} \frac{\log_2 n}{n}.$$

Constructing Access Patterns. For each execution, after UNVEIL generates I/O access traces for the sample, it sorts the I/O access requests based on file names and request timestamps. This allows the system to extract the I/O access sequence for each file in a given run, and check which processes accessed each file. The key idea is that after sorting the I/O access requests per file, repetition can be observed in the way I/O requests are generated on behalf of the malicious process.

The particular detection criterion used by the system to detect ransomware samples is to identify write and delete operations in I/O sequences in each malware run. In a successful ransomware attack, the malicious process typically aims to encrypt, overwrite, or delete user files at some point during the attack. In UNVEIL, these I/O

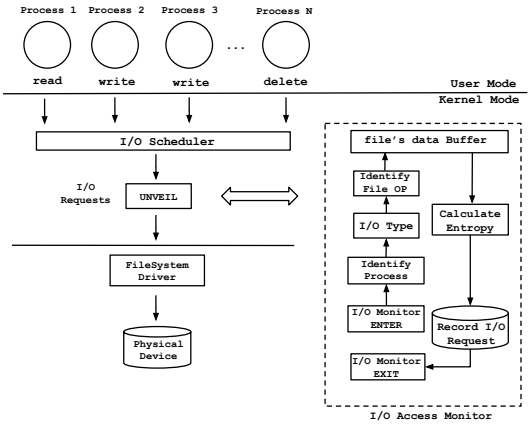

Figure 1: Overview of the design of I/O access monitor in UNVEIL. The module monitors system-wide filesystem accesses of user-mode processes. This allows UNVEIL to have full visibility into interactions with user files.

request patterns raise an alarm, and are detected as suspicious filesystem activity. We studied different file locker ransomware samples across different ransomware families. Our analysis shows that although these attacks can be very different in their attack strategies (e.g., evasion techniques, key generation, key management, connecting to C&C servers), they can be categorized into three main classes of attacks based on their access requests.

Figure 2 shows the high-level access patterns for multiple ransomware families we studied during our experiments. For example, the access pattern shown to the left is indicative of Cryptolocker variants that have varying key lengths and desktop locking techniques. However, its access pattern remains constant with respect to family variants. We observed the same I/O activity for samples in the CryptoWall family as well. While these families are identified as two different ransomware families, since they use the same encryption functions to encrypt files (i.e., the Microsoft CryptoAPI), they have similar I/O patterns when they attack user files.

As another example, in FileCoder family, the ransomware first creates a new file, reads data from a victim's file, generates an encrypted version of the original data, writes the encrypted data buffer to the newly generated file, and simply unlinks the original user's file (See Figure 2.2). In this class of file locker ransomware, the malware does not wipe the original file's data from the disk. For attack approaches like this, victims have a high chance of recovering their data without paying the ransom. In the third approach (Figure 2.3), however, the ransomware creates a new encrypted file based on the original file's data and then securely deletes the original file's data using either standard Windows APIs or custom overwriting implementations (e.g., such as Cryp-

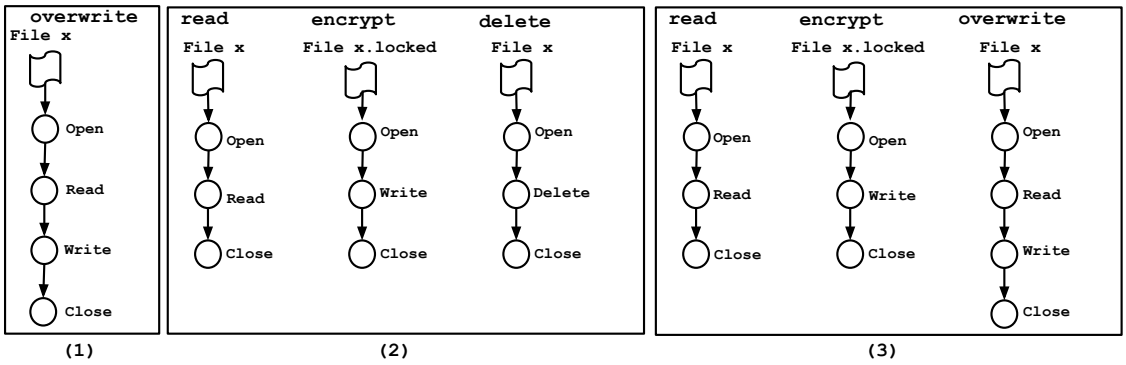

Figure 2: Strategies differ across ransomware families with respect to I/O access patterns. (1) Attacker overwrites the users' file with an encrypted version; (2) Attacker reads, encrypts and deletes files without wiping them from storage; (3) Attacker reads, creates a new encrypted version, and securely deletes the original files by overwriting the content.

Vault family).

3.2 Detecting Screen Lockers

The second core component of UNVEIL is aimed at detecting screen locker ransomware. The key insight behind this component is that the attacker must display a ransom note to the victim in order to receive a payment. In most cases, the message is prominently displayed, covering a significant part, or all, of the display. As this ransom note is a virtual invariant of ransomware attacks, UNVEIL aims to automatically detect the display of such notes.

The approach adopted by UNVEIL to detect screen locking ransomware is to monitor the desktop of the victim machine, and to attempt to detect the display of a ransom note. Similar to Grier et al. [15], we take automatic screenshots of the analysis desktop before and after the sample is executed. The screenshots are captured from outside of the dynamic analysis environment to prevent potential tampering by the malware. This series of screenshots is analyzed and compared using image analysis methods to determine if a large part of the screen has suddenly changed between captures. However, smaller changes in the image such as the location of the mouse pointer, current date and time, new desktop icons, windows, and visual changes in the task bar should be rejected as inconsequential.

In UNVEIL, we measure the *structural similarity* (SSIM) [49] of two screenshots – before and after sample execution – by comparing local patterns of pixel intensities in terms of both luminance and contrast as well as the structure of the two images. Extracting structural information is based on the observation that pixels have strong inter-dependencies – especially when they are spatially close. These dependencies carry information about the structure of the objects in the image. After a successful ransomware attack, the display of the ransom note often results in automatically identifiable changes in the structural information of the screenshot (e.g., a large rectangular object covers a large part of the desktop). Therefore, the similarity of the pre- and post-attack images decreases significantly, and can be used as an indication of ransomware.

In order to avoid false positives, UNVEIL only takes screenshots resulting from persistent changes (i.e., changes that cannot be easily dismissed through user interaction). The system first removes such transient changes (e.g., by automatically closing open windows) before taking screenshots of the desktop. Using this preprocessing step, ransomware-like applications that are developed for other purposes such as fake AV are safely categorized as non-ransomware samples.

UNVEIL also extracts the text within the area where changes in the structure of the image has occurred. The system extracts the text inside the selected area and searches for specific keywords that are highly correlated with ransom notes (e.g.,`<lock, encrypt, desktop, decryption, key>`).

Given two screenshots X and Y, we define the structural similarity index of the image contents of local windows x_j and y_j as

$$\mathsf{LocalSim}(x_j, y_j) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where μ_x and μ_y are the mean intensity of x_j and y_j, and σ_x and σ_y are the standard deviation as an estimate of x_j and y_j contrast and σ_{xy} is the covariance of x_j and y_j. The local window size to compare the content of two images was set 8×8. c_1 and c_2 are division stabilizer in the SSIM index formula [49]. We define the overall similarity between the two screenshots X and Y as the arithmetic mean of the similarity of the image contents x_j and y_j at the j^{th} local window where M is the number

of local windows of X and Y:

$$\mathsf{ImgSim}(X,Y) = \frac{1}{M}\sum_{j=1}^{M}\mathsf{LocalSim}(x_j,y_j).$$

Since the overall similarity is always on $[0,1]$, the distance between X and Y is simply defined as

$$\mathsf{Dist}(X,Y) = 1 - \mathsf{ImgSim}(X,Y).$$

Finally, we define a similarity threshold τ_{sim} such that UNVEIL considers the sample a potential screen locking ransomware if

$$\mathsf{Dist}(X,Y) > \tau_{\mathsf{sim}}.$$

UNVEIL then extracts the text within the image and searches for ransomware-related words within the modified area. Applying the image similarity test with the best similarity threshold (see Section 5.2.2) gives us the highest recall with 100% precision for the entire dataset.

4 UNVEIL Implementation

In this section, we describe the implementation details of a prototype of UNVEIL for the Windows platform. We chose Windows for a proof-of-concept implementation because it is currently the main target of ransomware attacks. We elaborate on how UNVEIL automatically generates artificial, but realistic user environments for each analysis run, how the system-wide monitoring was implemented, and how we deployed the prototype of our system.

4.1 Generating User Environments

In each run, the user environment is made up of several forms of content such as digital images, videos, audio files, and documents that can be accessed during a user *Windows Session*. The user content is automatically-generated according to the following process:

For each file extension from a space of possible extensions, a set of files are generated where the number of files for each extension is sampled from a uniform random distribution for each sample execution. Each set of files collectively forms a *document space* for the sample execution environment. From a statistical perspective, document spaces generated for each sample execution should be indistinguishable from real user data. As an approximation to this ideal, randomly-selected numbers of files are generated per extension for each run according to the process described above.

In the following, we describe the additional properties that a document space should have in order to complicate programmatic approaches that ransomware samples can potentially use to identify the automatically-generated user environment.

Valid Content. The user content generator creates real files with valid headers and content using standard libraries (e.g., python-docx, python-pptx, OpenSSL). Based on empirical observation, we created four file categories that a typical ransomware sample tries to find and encrypt: documents, keys and licenses, file archives, and media. Document extensions include `txt`, `doc(x)`, `ppt(x)`, `tex`, `xls(x)`, `c`, `pdf` and `py`. Keys and license extensions include `key`, `pem`, `crt`, and `cer`. Archive extensions include `zip` and `rar` files. Finally, media extensions include `jp(e)g`, `mp3`, and `avi`. For each sample execution, a subset of extensions are randomly selected and are used to generate user content across the system.

In order to generate content that appears meaningful, we collected approximately 100,000 sentences by querying 500 English words in Google. For each query, we collected the text from the first 30 search results to create a sentence list. We use the collected sentences to generate the content for the user files. We used the same technique to create a word list to give a name to the user files. The word list allows us to create files with variable name lengths that do not appear random. Clearly, the problem with random content and name generation (e.g., `xteyshtfqb.docx`) is that the attacker could programmatically calculate the entropy of the file names and contents to detect content that has been generated automatically. Hence, by generating content that appears meaningful, we make it difficult for the attacker to fingerprint the system and detect our generated files.

File Paths. Note that the system is also careful to randomly generate the supposed victim's directory structure. For example, directory names are also generated based on meaningful words. Furthermore, the system also associates files of certain types with standard locations in the Windows directory structure for those file types (e.g., the system does not create document files in a directory with image files, but rather under My Documents). The path length of user files is also non-deterministic and is generated randomly. In addition, each folder may have a set of sub-folders. Consequently, the generated paths to user files have variable depths relative to the root folder.

Time Attributes. Another non-determinism strategy used by our approach is to generate files with different *creation*, *modification*, and *access* times. The file time attributes are sampled from a distribution of likely timestamps when creating the file. When the system creates files with different time attributes, the time attributes of the containing folders are also updated automatically. In this case, the creation time of the folder is the minimum of all creation times of files and folders inside the folder, while the modification and access times are the maxi-

mum of the corresponding timestamps.

While we have not observed ransomware samples that have attempted to use fingerprinting heuristics of the content of the analysis environment, the nondeterminism strategies used by UNVEIL serve as a basis for making the analysis resilient to fingerprinting by design.

4.2 Filesystem Activity Monitor

Several techniques have been used to monitor sample filesystem activity in malware analysis environments. For example, filesystem activity can be monitored by hooking a list of relevant filesystem API functions or relevant system calls using the System Service Descriptor Table (SSDT). Unfortunately, these approaches are not suitable for UNVEIL's detection approach for several reasons. First, API hooking can be bypassed by simply copying a DLL containing the desired code and dynamically loading it into the process' address space under a different name. Stolen code [17, 19] and sliding calls [19] are other examples of API hooking evasion that are common in the wild. Furthermore, ransomware can use customized cryptosystems instead of the standard APIs to bypass API hooking while encrypting user files. Hooking system calls via the SSDT also has other technical limitations. For example, it is prevented on 64-bit systems due to Kernel Patch Protection (KPP). Furthermore, most SSDT functions are undocumented and subject to change across different versions of Windows.

Therefore, instead of API or system call hooking, UNVEIL monitors filesystem I/O activity using the Windows Filesystem Minifilter Driver framework [34], which is a standard kernel-based approach to achieving system-wide filesystem monitoring in multiple versions of Windows. The prototype consists of two main components for I/O monitoring and retrieving logs of the entire system with approximately 2,800 SLOC in C++. In Windows, I/O requests are represented by I/O Request Packets (IRPs). UNVEIL's monitor sets callbacks on all I/O requests to the filesystem generated on behalf of user-mode processes. Basing UNVEIL's filesystem monitor on a minifilter driver allows it to be located at the closest possible layer to the filesystem with access to nearly all objects of the operating system.

4.3 Desktop Lock Monitor

To identify desktop locking ransomware, screenshots are captured from outside of the dynamic analysis environment to prevent potential tampering by the malware. For dissimilarity testing, a python script implements the Structural Similarity Image Metric (SSIM) as described in Section 3.2. UNVEIL first converts the images to floating point data, and then calculates parameters such as mean intensity μ using Gaussian filtering of the images' contents. We also used default values ($k_1 = 0.01$ and $k_2 = 0.03$) to obtain the values of c_1 and c_2 to calculate the structural similarity score in local windows presented in Section 3.2.

The system also employs `Tesseract-OCR` [38], an open source OCR engine, to extract text from the selected areas of the screenshots. To perform the analysis on the extracted text within images, we collected more than 10,000 unique ransom notes from different ransomware families. We first clustered ransom notes based on the family type and the visual appearance of the ransom notes. For each cluster, we then extracted the ransom texts in the corresponding ransom notes and performed pre-filtering to remove unnecessary words within the text (e.g., articles, pronouns) to avoid obvious false positive cases. The result is a word list for each family cluster that can be used to identify ransom notes and furthermore label notes belonging to a known ransomware family.

5 Evaluation

We evaluated UNVEIL with two experiments. The goal of the first experiment is to demonstrate that the system can detect known ransomware samples, while the goal of the second experiment is to demonstrate that UNVEIL can detect previously unknown ransomware samples.

5.1 Experimental Setup

The UNVEIL prototype is built on top of Cuckoo Sandbox [13]. Cuckoo provides basic services such as sample submission, managing multiple VMs, and performing simple human interaction tasks such as simulating user input during an analysis. However, in principle, UNVEIL could be implemented using any dynamic analysis system (e.g., BitBlaze [5], VxStream Sandbox [37]).

We evaluated UNVEIL using 56 VMs running Windows XP SP3 on a Ganeti cluster based on Ubuntu 14.04 LTS. While Windows XP is not required by UNVEIL, it was chosen because it is well-supported by Cuckoo sandbox. Each VM had multiple NTFS drives. We took anti-evasion measures against popular tricks such as changing the IP address range and the MAC addresses of the VMs to prevent the VMs from being fingerprinted by malware authors. Furthermore, we permitted controlled access to the Internet via a filtered host-only adapter. In particular, the filtering allowed limited IRC, DNS, and HTTP traffic so samples could communicate with C&C servers. SMTP traffic was redirected to a local honeypot to prevent spam, and network bandwidth was limited to mitigate potential DoS attacks.

Family	Type	Samples
Cryptolocker	crypto	33 (1.5%)
CryptoWall	crypto	42 (2.0%)
CTB-Locker	crypto	77 (3.6%)
CrypVault	crypto	21 (1.0%)
CoinVault	crypto	17 (0.8%)
Filecoder	crypto	19 (0.9%)
TeslaCrypt	crypto	39 (1.8%)
Tox	crypto	71 (3.3%)
VirLock	locker	67 (3.2%)
Reveton	locker	501 (23.6%)
Tobfy	locker	357 (16.8%)
Urausy	locker	877 (41.3%)
Total Samples	-	**2,121**

Table 1: The list of ransomware families used in the first experiment.

The operating system image inside the malware analysis system included typical user data such as saved social networking credentials and a valid browsing history. For each operating system image, multiple users were defined to run the experiments. We also ran a script that emulated basic user activity while the malware sample was running on the system, such as launching a browser and navigating to multiple websites, or clicking on the desktop. This interaction was randomly-generated, but was constant across runs. Each sample was executed in the analysis environment for 20 minutes. As described in Sections 3.1 and 3.2, user environments were generated for each run, filesystem I/O traces were recorded, and pre- and post-execution screenshots were captured. After each execution, the VM was rolled back to a clean state to prevent any interference across executions. All experiments were performed according to well-established experimental guidelines [40] for malware experiments.

Run	OP	Proc	FName	Offset	Entropy
CryptoWall 3	`read`	`explorer.exe`	`document.cad`	[0,4096)	5.21
	`write`	`explorer.exe`	`document.cad`	[0,4096)	7.04
	...				
CryptoWall 4	`read`	`explorer.exe`	`project.cad`	[0,4096)	5.21
	`write`	`explorer.exe`	`project.cad`	[0,4096)	7.11
	...				
	`rename`	`explorer.exe`	`t67djkje.elkd8`		

Table 2: An example of I/O access in UNVEIL for CryptoWall 3.0 and CryptoWall 4.0.

Application	OP	Description
CrypVault	`read`	read low entropy buffer from original file
	`write`	write high entropy buffer to a new file
	...	
	`write`	overwrite the buffer of the original file
	`delete`	read attributes, delete the original file
CryptoWall4	`read`	read low entropy buffer
	`write`	overwrite with high entropy buffer
	...	
	`rename`	read attributes, rename the files
SDelete	`write`	overwrite data buffer
	...	
	`delete`	read attributes, delete the file
7-zip	`read`	read data buffer from original file
	`write`	write data buffer to a new file
	...	

Table 3: I/O accesses for deletion and compression mechanisms in benign/malicious applications. Benign programs can generate I/O access requests similar to ransomware attacks, but since they are not designed to deny access to the original files, their I/O sequence patterns are different from ransomware attacks.

5.2 Ground Truth (Labeled) Dataset

In this experiment, we evaluated the effectiveness of UNVEIL on a labeled dataset, and ran different screen locker samples to determine the best threshold value τ_{sim} for the large-scale experiment.

We collected ransomware samples from public repositories [1, 3] and online forums that share malware samples [2, 32]. We also received labeled ransomware samples from two well-known anti-malware companies. In total, we collected 3,156 recent samples. In order to make sure that those samples were indeed active ransomware, we ran them in our test environment. We confirmed 2,121 samples to be active ransomware instances. After each run, we checked the filesystem activity of each sample for any signs of attacks on user data. If we did not see any malicious filesystem activity, we checked whether running the sample displayed a ransom note.

Table 1 describes the ransomware families we used in this experiment. We note that the dataset covers the majority of the current ransomware families in the wild. In addition to the labeled ransomware dataset, we also created a dataset that consisted of non-ransomware samples. These samples were submitted to the Anubis analysis platform [16], and consisted of a collection of benign as well as malicious samples. We selected 149 benign executables including applications that have ransomware-like behavior such as secure deletion, encryption, and compression. A short list of these applications are provided in Table 5. We also tested 384 non-ransomware malware samples from 36 malware families to evaluate the false positive rate of UNVEIL. Table 2 shows an example of I/O traces for CryptoWall 3.0 and CryptoWall 4.0 where the victim's file is first read and then overwritten with an encrypted version. The I/O access patterns of CryptoWall 4.0 samples to overwrite the content of the files are identical since they use the same cryptosystem. The main difference is that the filenames and extensions are modified with random characters, probably to minimize the chance of recovering the files based on their names in the Master File Table (MFT) in the NTFS filesystem.

5.2.1 Filesystem Activity of Benign Applications with Potential Ransomware-like Behavior

One question that arises is whether benign applications such as encryption or compression programs might generate similar I/O request sequences, resulting in false positives. Note that with benign applications, the original file content is treated carefully since the ultimate goal is to generate an encrypted version of the original file, and not to restrict access to the file. Therefore, the default mechanism in these applications is that the original files remain intact even after encryption or compression. If automatic deletion is deliberately activated by the user after the encryption, it can potentially result in a false positive (see Figure 2.2). However, in our approach, we assume that the usual default behavior is exhibited and the original data is preserved. We believe that this is a reasonable assumption, considering that we are building an analysis system that will mainly analyze potentially suspicious samples captured and submitted for analysis. Nevertheless, we investigated the I/O access patterns of benign programs, shown in Table 3. The I/O traces indicate that these programs exhibit distinguishable I/O access patterns as a result of their default behavior.

Benign applications might not necessarily perform encryption or deletion on user files, but can change the content of the files. For example, updating the content of a Microsoft PowerPoint file (e.g., embedding images and media) generates I/O requests similar to ransomware (see Figure 2.1). However, the key difference here is that such applications usually generate I/O requests for a single file at a time and repetition of I/O requests does not occur over multiple user files. Also, note that benign applications typically do not arbitrarily encrypt, compress or modify user files, but rather need sophisticated input from users (e.g., file names, keys, options, etc.). Hence, most applications would simply exit, or expect some input when executed in UNVEIL.

5.2.2 Similarity Threshold

We performed a precision-recall analysis to find the best similarity threshold τ_{sim} for desktop locking detection. The best threshold value to discriminate between similar and dissimilar screenshots should be defined in such a way that UNVEIL is be able to detect screen locker ransomware while maintaining an optimal precision-recall rate. Figure 3 shows empirical precision-recall results when varying τ_{sim}. As the figure shows, with $\tau_{\text{sim}} = 0.32$, more than 97% of the ransomware samples across both screen and file locker samples are detected with 100% precision. In the second experiment, we used this similarity threshold to detect screen locker ransomware in a malware feed unknown to UNVEIL.

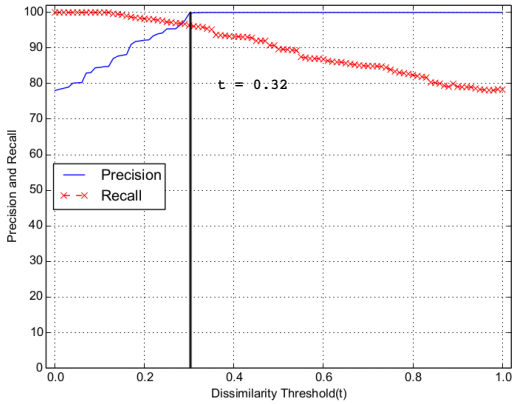

Figure 3: Precision-recall analysis of the tool. The threshold value $\tau_{\text{sim}} = 0.32$ gives the highest recall with 100% precision.

Evaluation	Results
Total Samples	148,223
Detected Ransomware	13,637 (9.2%)
Detection Rate	96.3%
False Positives	0.0%
New Detection	9,872 (72.2%)

Table 4: UNVEIL detection results. 72.2% of the ransomware samples detected by UNVEIL were not detected by any of AV scanners in VirusTotal at the time of the first submission. 7,572 (76.7%) of the newly detected samples were destructive file locker ransomware samples.

5.3 Detecting Zero-Day Ransomware

The main goal of the second experiment is to evaluate the accuracy of UNVEIL when applied to a large dataset of recent real-world malware samples. We then compared our detection results with those reported by AV scanners in VirusTotal.

This dataset was acquired from the daily malware feed provided by Anubis [16] to security researchers. The samples were collected from May 18th 2015 until February 12th 2016. The feed is generated from the Anubis submission queue, which is fed in turn by Internet users and security companies. Hence, before performing the experiment, we filtered the incoming Anubis samples by removing those that were not obviously executable (e.g., PDFs, images). After this filtering step, the dataset contained 148,223 distinct samples. Each sample was then submitted to UNVEIL to obtain I/O access traces and pre-/post-execution desktop image dissimilarity scores.

5.3.1 Detection Results

Table 4 shows the evaluation results of the second experiment. With the similarity threshold $\tau_{\text{sim}} = 0.32$, UNVEIL labeled 13,637 (9.2% of the dataset) samples in the Anu-

bis malware feed as being ransomware; these included both file locker and desktop locker samples.

Evaluation of False Positives. As we did not have a labeled ground truth in the second experiment, we cannot provide an accurate precision-recall analysis, and verifying the detection results is clearly challenging. For example, re-running samples while checking for false positives is not feasible in all cases since samples may have become inactive at the time of re-analysis (e.g., the C&C server might have been taken down).

Hence, we used manual verification of the detection results. That is, for the samples that were detected as screen locker ransomware, we manually checked the post-attack screenshots that were reported taken by UNVEIL. The combination of structural similarity test and the OCR technique to extract the text provides a reliable automatic detection for this class of ransomware. We confirmed that UNVEIL correctly reported 4,936 samples that delivered a ransom note during the analysis.

Recall that UNVEIL reports a sample as a file locker ransomware if the I/O access pattern follows one of the three classes of ransomware attacks described in Figure 2. For file locker ransomware samples, we used the I/O reports for each sample. We listed all the I/O activities on the first five user files during that run and looked for suspicious I/O activity such as requesting *write* and/or *delete* operations. Note that the detection approach used in UNVEIL is only based on the I/O access pattern. We do not check for changes in entropy in the detection phase and it is only used for our evaluation.

If we find multiple write or delete I/O requests to the first five generated user files and also a significant increase in the entropy between read and write data buffers at a given file offset, or the creation of new high entropy files, we confirmed the detection as a true positive. The creation of multiple new high entropy files based on user files is a reliable sign of ransomware in our tests. For example, the malware sample that uses secure deletion techniques may overwrite files with low entropy data. However, the malicious program first needs to generate an encrypted version of the original files. In any case, generating high entropy data raises an alarm in our evaluation.

By employing these two approaches and analyzing the results, we did not find any false positives. There were a few cases that had significant change in the structure of the images. Our closer investigation revealed that the installed program generated a large installation page, showed some unreadable characters in the window, and did not close even if the button was clicked (i.e., non-functional buttons). In another case, the program generated a large setup window, but it did not proceed due to a crash. These cases produce a higher dissimilarity score than the threshold value. However, since the extracted text within those particular windows did not contain any ransomware-related contents, UNVEIL safely categorized them as being non-ransomware samples.

Evaluation of False Negatives. Determining false negative rates is a challenge since manually checking 148,223 samples is not feasible. In the following, we provide an *approximation* of false negatives for UNVEIL.

In our tests on the labeled dataset, false negatives mainly occurred in samples that make persistent changes on the desktop, but since the dissimilarity score of pre-/post-attack is less that $\tau_{\mathsf{sim}} = 0.32$, it is not detected as ransomware by UNVEIL. Our analysis of labeled samples from multiple ransomware families (see Section5.2) shows that these cases were mainly observed in samples with a similarity score between the interval [0.18, 0.32). This is because for lower similarity scores, changes in the screenshots are negligible or small (e.g., Windows warning/error messages). Consequently, in order to increase the chance of catching false negative cases, we selected all the samples where their dissimilarity score was between [0.18, 0.32). This decreases the size of potential desktop locker ransomware that were not detected by UNVEIL to 4,642 samples. We manually checked the post-attack screenshots of these samples, and found 377 desktop locker ransomware that UNVEIL was not able to detect. Our analysis shows that the false negatives in desktop locker ransomware resulted from samples in one ransomware family that generated a very transparent ransom note with a dissimilarity score between [0.27, 0.31] that was difficult to read.

For file locker ransomware, we first removed the samples that were not detected as malware by any of the AV scanners in VirusTotal after multiple resubmissions in consecutive days (see Section 5.3.2). By applying this approach, we were able to reduce the number of samples to check by 47%. Then, we applied a similar approach we used as described above. We listed the first five user files generated for that sample run and checked whether any process requested write access to those files. We also checked the entropy of multiple data buffers. If we identified write access with a significant increase in the entropy of data buffers compared to the entropy of data buffer in the read access for those files, we report it as a false negative.

Our test shows that UNVEIL does not have any false negatives in file locker ransomware samples. Consequently, we conclude that UNVEIL is able to detect multiple classes of ransomware attacks with a low false positive rate (FPs = 0.0% at a TP = 96.3%).

5.3.2 Early Warning

One of the design goals of UNVEIL is to be able to automatically detect previously unknown (i.e., zero-day) ransomware. In order to run this experiment, we did the following. Once per day over the course of the experiment, we built a malware dataset that was concurrently submitted to UNVEIL and VirusTotal. If a sample was detected as ransomware by UNVEIL, we checked the VirusTotal (VT) detection results. In cases where a ransomware sample was not detected by any VT scanner, we reported it as a new detection.

In addition, we also measured the lag between a new detection by UNVEIL and a VT detection. To that end, we created a dataset from the newly detected samples submitted on days $\{1,2,\ldots,n-1,n\}$ and resubmitted these samples to see whether the detection results changed. We considered the result of all 55 VT scanners in this experiment. Since the number of scanners is relatively high, we defined a VT detection ratio ρ as the ratio of the total number of scanners that identified the sample as ransomware or malware to the total number of scanners checked by VT. ρ is therefore a value on the interval [0,1] where zero means that the sample was not detected by any of the 55 VT scanners, and 1 means that all scanners reported the sample as malware or ransomware. Since there is no standard labeling scheme for malware in the AV industry, a scanner can label a sample using a completely different name from another scanner. Consequently, to avoid biased results, we consider the labeling of a sample using *any* name as a successful detection.

In our experiment, we submitted the detected samples every day to see how the VT detection ratio ρ changes over time. The distribution of ρ for each submission is shown in Figure 4. Our analysis shows that ρ does not significantly change after a small number of subsequent submissions. For the first submission, 72.2% of the ransomware samples detected by UNVEIL were not detected by any of the 55 VT scanners. After a few submissions, ρ does not change significantly, but generally was concentrated either towards small or very large ratios. This means that after a few re-submissions, either only a few scanners detected a sample, or almost all the scanners detected the sample.

5.4 Case Study: Automated Detection of a New Ransomware Family

In this section, we describe a new ransomware family, called SilentCrypt, that was detected by UNVEIL during the experiments. After our system detected these samples and submitted them to VirusTotal, several AV vendors picked up on them and also started detecting them a couple of days later, confirming the malice of the sample that we automatically detected.

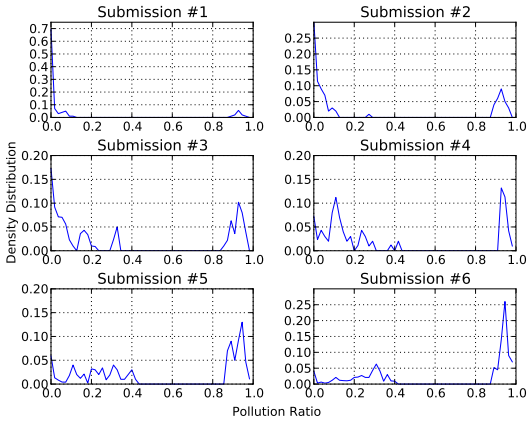

Figure 4: Evolution of VT scanner reports after six submissions. 72.2% of the samples detected by UNVEIL were not detected by any of AV scanners in the first submission. After a few re-submissions, the detection results do not change significantly. The detection results tend to be concentrated either towards small or very large detection ratios. This means that a sample is either detected by a relatively small number of scanners, or almost all of the scanners.

This family uses a unique and effective method to fingerprint the runtime environment of the analysis system. Unlike other malware samples that check for specific artifacts such as registry keys, background processes, or platform-specific characteristics, this family checks the private files of a user to determine if the code is running in an analysis environment. When the sample is executed, it first checks the number of files in the user's directories, and sends this list to the C&C server before starting the attack.

Multiple online malware analysis systems such as malwr.com, Anubis, and a modern sandboxing technology provided by a well-known, anti-malware company did not register any malicious activity for this sample. However, the sample showed heavy encryption activity when analyzed by UNVEIL.

An analysis of the I/O activity of this sample revealed that this family first waited for several minutes before attacking the victim's files. Figure 5 shows the three main I/O activities of one of the samples in this family. The sample traverses the current user's main directories, and creates a list of files and folders. If the sample receives permission to attack from the C&C server, it begins encrypting the targeted files. To confirm UNVEIL's alerts, we conducted a manual investigation over several days. Our analysis concluded that the malicious activity is started only if user activity is detected. Unlike other ransomware samples that imme-

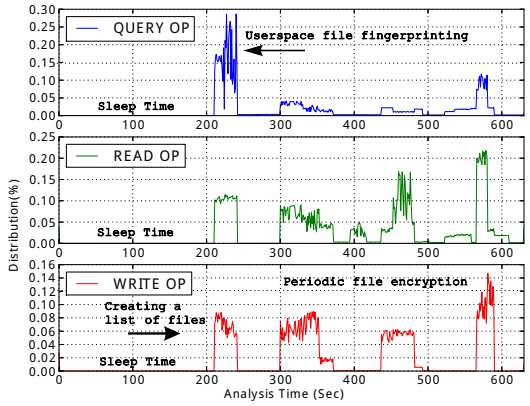

Figure 5: I/O activities of a previously unknown ransomware family detected by UNVEIL. The sample first performs victim file fingerprinting to ensure that the running environment is not a bare user environment.

diately attack a victim's files when they are executed, this family only encrypt files that have recently been opened by the user while the malicious process is monitoring the environment. That is, the malicious process reads the file's data and overwrites it with encrypted data if the file is used. The file name is then updated to `"filename.extension.locked_forever"` after it has been encrypted.

UNVEIL was able to detect this family of ransomware automatically because it was triggered after the system accessed some of the generated user files as a part of the user activity emulation scripts. Once we submitted the sample to VirusTotal, the sample was picked up by other AV vendors (5/55) after five days with different labels. A well-known, sandboxing-based security company confirmed our findings that the malware sample was a new threat that they had not detected before. We provide an anonymous video of a sample from this ransomware family in [6].

6 Discussion and Limitations

The evaluation in Section 5 demonstrates that UNVEIL achieves good, practical, and useful detection results on a large, real-world dataset. Unfortunately, malware authors continuously observe defensive advances and adapt their attacks accordingly. In the following, we discuss limitations of UNVEIL and potential evasion strategies.

There is always the possibility that attackers will find ways to fingerprint the automatically generated user environment and avoid it. However, this comes at a high cost, and increases the difficulty bar for the attacker. For example, in desktop-locking ransomware, malware can use heuristics to look for specific user interaction before locking the desktop (e.g., waiting for multiple login events or counting the number of user clicks). However, implementing these approaches can potentially make detection easier since these approaches require hooking specific functions in the operating system. The presence of these hooking behaviors are themselves suspicious and are used by current malware analysis systems to detect different classes of malware. Furthermore, these approaches delay launching the attack which increases the risk of being detected by AV scanners on clients before a successful attack occurs.

Another possibility is that a malware might only encrypt a specific part of a file instead of aggressively encrypting the entire file, or simply shuffle the file content using a specific pattern that makes the files unreadable. Although we have not seen any sample with these behaviors, developing such ransomware is quite possible. The key idea is that in order to perform such activities, the malicious program should open the file with write permission and manipulate at least some data buffers of the file content. In any case, if the malicious program accesses the files, UNVEIL will still see this activity. There is no real reason for benign software to touch automatically generated files with write permission and modify the content. Consequently, such activities will still be logged. Malware authors might use other techniques to notify the victim and also evade the desktop lock monitor. As an example, the ransomware may display the ransom note via video or audio files rather than locking the desktop. As we partially discussed, these approaches only make sense if the malware is able to successfully encrypt user files first. In this case, UNVEIL can identify those malicious filesystem access as discussed in Section 3.1.2.

We also believe that the current implementation of text extraction to detect desktop locker ransomware can be improved. We observed that the change in the structure of the desktop screen-shots is enough to detect a large number of current ransomware attacks since UNVEIL exploits the attacker's goal which is to ensure that the victims see the ransom note. However, we believe that the text extraction module can be improved to detect possible evasion techniques an attacker could use to generate the ransom note (e.g., using uncommon words in the ransom text).

Clearly, there is always the possibility that an attacker will be able to fingerprint the dynamic analysis environment. For example, stalling code [26] has become increasingly popular to prevent the dynamic analysis of a sample. Such code takes longer to execute in a virtual environment, preventing execution from completing during an analysis. Also, attackers can actively look for signs of dynamic analysis (e.g., signs of execution in a VM

such as well-known hard disk names). Note that UNVEIL is agnostic as to the underlying dynamic analysis environment. Hence, as a mitigation, UNVEIL can use a sandbox that is more resistant to these evasion techniques(e.g., [26, 48]). The main contribution of UNVEIL is not the dynamic analysis of malware, but rather the introduction of new techniques for the automated, specific detection of ransomware during dynamic analysis.

UNVEIL runs within the kernel, and aims to detect user-level ransomware. As a result, there is the risk that ransomware may run at the kernel level and thwart some of the hooks UNVEIL uses to monitor the filesystem. However, this would require the ransomware to run with administrator privileges to load kernel code or exploit a kernel vulnerability. Currently, most ransomware runs as user-level programs because this is sufficient to carry out ransomware attacks. Kernel-level attacks would require more sophistication, and would increase the difficulty bar for the attackers. Also, if additional resilience is required, the kernel component of UNVEIL could be moved outside of the analysis sandbox.

7 Related Work

Many approaches have been proposed to date that have aimed to improve the analysis and detection of malware. A number of approaches have been proposed to describe program behavior from analyzing byte patterns [29, 43, 41, 50] to transparently running programs in malware analysis systems [4, 23, 22, 47]. Early steps to analyze and capture the main intent of a program focused on analysis of control flow. For example, Kruegel et al. [28] and Bruschi et al. [9] showed that by modeling programs based on their instruction-level control flow, it is possible to bypass some forms of obfuscation. Similarly, Christodorescu et al. [12] used instruction-level control flow to design obfuscation-resilient detection systems. Later work focused on analyzing and detecting malware using higher-level semantic characterizations of their runtime behavior derived from sequences of system call invocations and OS resource accesses [24, 25, 11, 33, 42, 51].

Similar to our use of automatically-generated user content, decoys have been used in the past to detect security breaches. For instance, the use of decoy resources has been proposed to detect insider attacks [8, 52]. Recently, Juels et al. [18] used *honeywords* to improve the security of hashed passwords. The authors show that decoys can improve the security of hashed passwords since the attempt to use the decoy password for logins results in an alarm. In other work, Nikiforakis et al. [35] used decoy files to detect illegally obtained data from file hosting services.

There have also been some recent reports on the ransomware threat. For example, security vendors have reported on the threat of potential of ransomware attacks based on the number of infections that they have observed [46, 7, 44, 36]. A first report on specific ransomware families was made by Gazet where the author analyzed three ransomware families including `Krotten` and `Gpcode` [14]. The author concluded that while these early families were designed for massive propagation, they did not fulfill the basic requirements for mass extortion (e.g., sufficiently long encryption keys). Recently, Kharraz et al. [21] analyzed 15 ransomware families and provided an evolution-based study of ransomware attacks. They performed an analysis of charging methods and the use of Bitcoin for monetization. They proposed several high-level mitigation strategies such as the use of decoy resources to detect suspicious file access. Their assumption is that every filesystem access to delete or encrypt decoy resources is malicious and should be reported. However, they did not implement any concrete solution to detect or defend against these attacks.

We are not aware of any systems that have been proposed in the literature that specifically aim to detect ransomware in the wild. In particular, in contrast to existing work on generic malware detection, UNVEIL detects behavior specific to ransomware (e.g., desktop locking, patterns of filesystem accesses).

8 Conclusions

In this paper we presented UNVEIL, a novel approach to detecting and analyzing ransomware. Our system is the first in the literature to specifically identify typical behavior of ransomware such as malicious encryption of files and locking of user desktops. These are behaviors that are difficult for ransomware to hide or change.

The evaluation of UNVEIL shows that our approach was able to correctly detect 13,637 ransomware samples from multiple families in a real-world data feed with zero false positives. In fact, UNVEIL outperformed all existing AV scanners and a modern industrial sandboxing technology in detecting both superficial and technically sophisticated ransomware attacks. Among our findings was also a new ransomware family that no security company had previously detected before we submitted it to VirusTotal.

9 Acknowledgements

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

References

[1] Minotaur Analysis - Malware Repository. minotauranalysis.com.

[2] Malware Tips - Your Security Advisor. http://malwaretips.com/forums/viruses-exchange.104/.

[3] MalwareBlackList - Online Repository of Malicious URLs. http://www.malwareblacklist.com.

[4] Proof-of-concept Automated Baremetal Malware Analysis Framework. https://code.google.com/p/nvmtrace/.

[5] BitBlaze Malware Analysis Service. http://bitblaze.cs.berkeley.edu/, 2016.

[6] SilentCrypt: A new ransomware family. https://www.youtube.com/watch?v=qiASKA4BMck, 2016.

[7] AJJAN, A. Ransomware: Next-Generation Fake Antivirus. http://www.sophos.com/en-us/medialibrary/PDFs/technicalpapers/SophosRansomwareFakeAntivirus.pdf, 2013.

[8] BOWEN, B. M., HERSHKOP, S., KEROMYTIS, A. D., AND STOLFO, S. J. *Baiting inside attackers using decoy documents*. Springer, 2009.

[9] BRUSCHI, D., MARTIGNONI, L., AND MONGA, M. Detecting self-mutating malware using control-flow graph matching. In *Detection of Intrusions and Malware & Vulnerability Assessment*. Springer, 2006, pp. 129–143.

[10] CATALIN CIMPANU. Breaking Bad Ransomware Completely Undetected by VirusTotal. http://http://news.softpedia.com/news/breaking-bad-ransomware-goes-completely-undetected-by-virustotal-493265.shtml, 2015.

[11] CHRISTODORESCU, M., JHA, S., AND KRUEGEL, C. Mining specifications of malicious behavior. In *Proceedings of the 1st India software engineering conference* (2008), ACM, pp. 5–14.

[12] CHRISTODORESCU, M., JHA, S., SESHIA, S. A., SONG, D., AND BRYANT, R. E. Semantics-aware malware detection. In *Security and Privacy, 2005 IEEE Symposium on* (2005), IEEE, pp. 32–46.

[13] CUCKOO FOUNDATION. Cuckoo Sandbox: Automated Malware Analysis. www.cuckoosandbox.org, 2015.

[14] GAZET, A. Comparative analysis of various ransomware virii. *Journal in Computer Virology 6*, 1 (February 2010), 77–90.

[15] GRIER, C., BALLARD, L., CABALLERO, J., CHACHRA, N., DIETRICH, C. J., LEVCHENKO, K., MAVROMMATIS, P., MCCOY, D., NAPPA, A., PITSILLIDIS, A., ET AL. Manufacturing compromise: the emergence of exploit-as-a-service. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), pp. 821–832.

[16] INTERNATIONAL SECURE SYSTEM LAB. Anubis - Malware Analysis for Unknown Binaries. https://anubis.iseclab.org/, 2015.

[17] JASHUA TULLY. An Anti-Reverse Engineering Guide. http://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide#StolenBytes, 2008.

[18] JUELS, A., AND RIVEST, R. L. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 145–160.

[19] KAWAKOYA, Y., IWAMURA, M., SHIOJI, E., AND HARIU, T. Api chaser: Anti-analysis resistant malware analyzer. In *Research in Attacks, Intrusions, and Defenses*. Springer, 2013, pp. 123–143.

[20] KEVIN SAVAGE, PETER COOGAN, HON LAU. the Evolution of Ransomware. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf, 2015.

[21] 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)* (07 2015).

[22] KIRAT, D., VIGNA, G., AND KRUEGEL, C. Barebox: efficient malware analysis on bare-metal. In *Proceedings of the 27th Annual Computer Security Applications Conference* (2011), ACM, pp. 403–412.

[23] KIRAT, D., VIGNA, G., AND KRUEGEL, C. Barecloud: Bare-metal analysis-based evasive malware detection. In *23rd USENIX Security Symposium (USENIX Security 14)* (2014), USENIX Association, pp. 287–301.

[24] KIRDA, E., KRUEGEL, C., BANKS, G., VIGNA, G., AND KEMMERER, R. Behavior-based spyware detection. In *Usenix Security* (2006), vol. 6.

[25] KOLBITSCH, C., COMPARETTI, P. M., KRUEGEL, C., KIRDA, E., ZHOU, X.-Y., AND WANG, X. Effective and efficient malware detection at the end host. In *USENIX security symposium* (2009), pp. 351–366.

[26] KOLBITSCH, C., KIRDA, E., AND KRUEGEL, C. The power of procrastination: detection and mitigation of execution-stalling malicious code. In *Proceedings of the 18th ACM conference on Computer and communications security* (2011), ACM, pp. 285–296.

[27] KREBS, B. FBI: North Korea to Blame for Sony Hack. http://krebsonsecurity.com/2014/12/fbi-north-korea-to-blame-for-sony-hack/, 2014.

[28] KRUEGEL, C., KIRDA, E., MUTZ, D., ROBERTSON, W., AND VIGNA, G. Polymorphic worm detection using structural information of executables. In *Recent Advances in Intrusion Detection* (2006), Springer, pp. 207–226.

[29] LI, W.-J., WANG, K., STOLFO, S. J., AND HERZOG, B. Fileprints: Identifying file types by n-gram analysis. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC* (2005), IEEE, pp. 64–71.

[30] LIN, J. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory 37* (1991), 145–151.

[31] LINDORFER, M., KOLBITSCH, C., AND COMPARETTI, P. M. Detecting environment-sensitive malware. In *Recent Advances in Intrusion Detection* (2011), Springer, pp. 338–357.

[32] 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.

[33] MARTIGNONI, L., STINSON, E., FREDRIKSON, M., JHA, S., AND MITCHELL, J. C. A layered architecture for detecting malicious behaviors. In *Recent Advances in Intrusion Detection* (2008), Springer, pp. 78–97.

[34] MICROSOFT, INC. File System Minifilter Drivers. https://msdn.microsoft.com/en-us/library/windows/hardware/ff540402%28v=vs.85%29.aspx, 2014.

[35] NIKIFORAKIS, N., BALDUZZI, M., ACKER, S. V., JOOSEN, W., AND BALZAROTTI, D. Exposing the lack of privacy in file hosting services. In *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats (LEET)* (March 2011), LEET 11, USENIX Association.

[36] O'GORMAN, G., AND MCDONALD, G. Ransomware: A Growing Menance. http://www.symantec.com/connect/blogs/ransomware-growing-menace, 2012.

[37] PAYLOAD SECURITY INC,. Payload Security. https://www.hybrid-analysis.com, 2016.

[38] RAY SMITH. Tesseract Open Source OCR Engine . https://github.com/tesseract-ocr/tesseract, 2015.

[39] REAQTA INC,. HyraCrypt Ransomware. https://reaqta.com/2016/02/hydracrypt-ransomware/, 2016.

[40] ROSSOW, C., DIETRICH, C. J., GRIER, C., KREIBICH, C., PAXSON, V., POHLMANN, N., BOS, H., AND VAN STEEN, M. Prudent practices for designing malware experiments: Status quo and outlook. In *Security and Privacy (SP), 2012 IEEE Symposium on* (2012), IEEE, pp. 65–79.

[41] SCHULTZ, M. G., ESKIN, E., ZADOK, E., AND STOLFO, S. J. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on* (2001), IEEE, pp. 38–49.

[42] STINSON, E., AND MITCHELL, J. C. Characterizing bots remote control behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2007, pp. 89–108.

[43] SUNG, A. H., XU, J., CHAVEZ, P., AND MUKKAMALA, S. Static analyzer of vicious executables (save). In *Computer Security Applications Conference, 2004. 20th Annual* (2004), IEEE, pp. 326–334.

[44] SYMANTEC, INC. Internet Security Threat Report. http://www.symantec.com/security_response/publications/threatreport.jsp, 2014.

[45] THE CYBER THREAT ALLIANCE. Lucrative Ransomware Attacks: Analysis of Cryptowall Version 3 Threat. http://cyberthreatalliance.org/cryptowall-report.pdf, 2015.

[46] TRENDLABS. An Onslaught of Online Banking Malware and Ransomware. http://apac.trend

micro.com/cloud-content/apac/pdfs/security-intelligence/reports/rpt-cashing-in-on-digital-information.pdf, 2013.

[47] VASUDEVAN, A., AND YERRABALLI, R. Cobra: Fine-grained malware analysis using stealth localized-executions. In *Security and Privacy, 2006 IEEE Symposium on* (2006).

[48] VIGNA, G. From Anubis and Wepawet to Llama. http://info.lastline.com/blog/from-anubis-and-wepawet-to-llama, June 2014.

[49] WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13*, 4 (2004), 600–612.

[50] XU, J.-Y., SUNG, A. H., CHAVEZ, P., AND MUKKAMALA, S. Polymorphic malicious executable scanner by api sequence analysis. In *Hybrid Intelligent Systems, 2004. HIS'04. Fourth International Conference on* (2004), IEEE, pp. 378–383.

[51] YIN, H., SONG, D., EGELE, M., KRUEGEL, C., AND KIRDA, E. Panorama: capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, pp. 116–127.

[52] YUILL, J., ZAPPE, M., DENNING, D., AND FEER, F. Honeyfiles: deceptive files for intrusion detection. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC* (2004), IEEE, pp. 116–122.

A Benign Applications Used in Experiment One

Application	Main Capability	Version
7-zip	Compression	15.06
Winzip	Compression	19.5
WinRAR	Compression	5.21
DiskCryptor	Encryption	1.1.846.118
AESCrypt	Encryption	—
Eraser	Shredder	6.2.0.2969
SDelete	Shredder	1.61

Table 5: The list of benign applications that generate similar I/O access patterns to ransomware.