PayBreak : Defense Against Cryptographic Ransomware

Eugene Kolodenker[‡⋆], William Koch[⋆], Gianluca Stringhini[†], and Manuel Egele[⋆]
[⋆]Boston University, [‡]MITRE, [†]University College London
{eugenek, wfkoch, megele}@bu.edu, g.stringhini@ucl.ac.uk

ABSTRACT

Similar to criminals in the physical world, cyber-criminals use a variety of illegal and immoral means to achieve monetary gains. Recently, malware known as *ransomware* started to leverage strong cryptographic primitives to hold victims' computer files "hostage" until a ransom is paid. Victims, with no way to defend themselves, are often advised to simply pay. Existing defenses against ransomware rely on ad-hoc mitigations that target the incorrect use of cryptography rather than generic live protection. To fill this gap in the defender's arsenal, we describe the approach, prototype implementation, and evaluation of a novel, automated, and most importantly *proactive* defense mechanism against ransomware. Our prototype, called PayBreak, effectively combats ransomware, and keeps victims' files safe.

PayBreak is based on the insight that secure file encryption relies on hybrid encryption where symmetric session keys are used on the victim computer. PayBreak observes the use of these keys, holds them in escrow, and thus, can decrypt files that would otherwise only be recoverable by paying the ransom. Our prototype leverages low overhead dynamic hooking techniques and asymmetric encryption to realize the key escrow mechanism which allows victims to restore the files encrypted by ransomware. We evaluated PayBreak for its effectiveness against twenty hugely successful families of real-world ransomware, and demonstrate that our system can restore all files that are encrypted by samples from twelve of these families, including the infamous CryptoLocker, and more recent threats such as Locky and SamSam. Finally, PayBreak performs its protection task at negligible performance overhead for common office workloads and is thus ideally suited as a *proactive* online protection system.

Keywords

ransomware; malware; hybrid cryptosystem; key vault; cyber crime

ASIA CCS '17, April 02-06, 2017, Abu Dhabi, United Arab Emirates

DOI: http://dx.doi.org/10.1145/3052973.3053035

1. INTRODUCTION

The goal of cyber-criminals, akin to real world criminals, is commonly monetary gain. Thus, over recent years the underground economy developed a multitude of approaches that allow cyber-criminals to make money. Attackers sell exploit kits [21] to infect victims via drive-by-download attacks, they operate exploit-as-a-service schemes [30], establish and rent out botnets, and even offer off-the-shelf solutions to run spam campaigns including customer service. Besides such established enterprises, so-called *ransomware* (an amalgamation of the words ransom and malware) has been established as another vector to enhance the profits of cyber-criminals. Ransomware is malware that prevents the victim user's access to a valuable resource and extorts a ransom payment to reestablish access. Ransomware comes in many forms and shapes. This includes, for example, malicious screen lockers on mobile devices or crypto-based ransomware that encrypts the victim's file with state-of-the-art cryptographic algorithms. It is this crypto-based ransomware that we target in this work. Hence, all subsequent references to ransomware in this paper refer to the class of ransomware that uses cryptography to encrypt user files for ransom. As with many benign software projects, early versions of crypto-based ransomware were unsuccessful because the malware authors decided to ignore the popular adage of "don't roll your own crypto." For example, weak home-brew crypto was responsible for the short lifetime of the first wave of GPCode [38]. However, the malware authors quickly learned their lesson and increasingly employ strong cryptographic algorithms which lead the Federal Bureau of Investigation to state that "To be honest, we often advise people just to pay the ransom." [6].

Recently, ransomware has drawn the attention of the research community and security vendors alike. However, all existing systems that try to address the threat of ransomware do so reactively. That is, provided a sample, existing systems can identify and protect from similar samples. For example, Kharraz et al. [34] proposed a mechanism that executes malware in an instrumented environment and accurately identifies ransomware. Unfortunately, the proposed system only detects ransomware after the user's files have been encrypted. Thus, files that get encrypted before the ransomware is identified as such remain inaccessible to the user. Additionally, collaborations between law enforcement and security vendors have resulted in tools that can reverse the encryption performed by some ransomware samples [8]. While such efforts deserve applause, they are inherently non-

scalable as demonstrated by the plethora of successful ransomware families.

We deem the existing techniques that address the threat of ransomware reactively inadequate. Instead, we propose a system that allows conscientious users to proactively defend themselves from ransomware attacks. The key benefit of our system is that it allows victims to recover from ransomware infections without paying the ransom. To this end, we propose a key escrow mechanism that securely stores cryptographic encryption keys in a key vault. Key escrow systems are consistently rejected by the research community for very good reasons. For example, government-mandated key escrow would give government agencies access to the cryptographic keys, and thus the ability to infringe on the user's privacy by decrypting data that is believed to be cryptographically protected. The stark difference between such key escrow systems and the scheme proposed in this work is that in our system the user has *exclusive* access to the cryptographic keys stored in escrow.

In an initial step, the user must generate an asymmetric key pair and add the public key to the system. This public key is used to encrypt keys that are placed in the key vault. During normal operation, our system monitors the programs that execute on a system and intercepts calls to functions that implement cryptographic primitives. Furthermore, the system captures symmetric encryption keys, encrypts them with the public key, and stores the result in the key vault. Once the user gets infected with ransomware and learns that she should pay a ransom to get access to her files, she can simply decrypt the key vault with her private key and decrypt the files without making any payments.

We implemented PayBreak as a prototype of our approach for the common ransomware target, Windows 7, although no special features of the Windows 7 operating system are used in our system. Additionally, we evaluated its effectiveness against a set of 107 recent ransomware samples from 20 distinct families. Furthermore, we performed micro benchmarks to measure the performance impact of our instrumentation as well as macro benchmarks to measure the activity that the key vault would receive during regular office tasks. While the micro benchmarks result in significant overhead for symmetric crypto operations, the macro benchmarks indicate that cryptographic operations are rare enough that the overhead created by our system (4.1ms) is below the human perception threshold.

In summary, this paper makes the following novel contributions:

- We identify salient characteristics that allow us to mitigate the threat of modern crypto-based ransomware (§2).
- We propose a key vault mechanism that pro-actively protects against the threats posed by crypto-based ransomware (§3).
- We implemented the proposed mechanism in a system called PayBreak for the Windows 7 operating system (§4).
- We evaluate PayBreak by running 107 ransomware samples in a controlled environment and demonstrate that PayBreak successfully recovers all files encrypted by any of twelve active and economically hugely successful ransomware families (§5.2).
- We also assess the performance impact of PayBreak based on micro benchmarks that target our modifications specifically and macro benchmarks that capture the impact of PayBreak on more realistic workloads (§5.3).

2. BACKGROUND

In this section we first discuss the typical flow of modern ransomware and the practical limitations that affect the design space for ransomware authors. We also discuss the threat model that our system is designed to operate under.

2.1 Practical considerations for ransomware

The goal of ransomware is to deny victims access to their data and extort a ransom payment in exchange of reestablishing access. The authors of ransomware soon realized that cryptography provides a reliable way to run such an extortion racket. Broadly speaking, ransomware can choose between symmetric (i.e., secret key) and asymmetric (i.e., public key) cryptography. Early versions relied on symmetric encryption and the anti-malware community was quick in reverse engineering the malware and providing decryption tools [41]. This was only possible because in symmetric encryption schemes the same key must be used to encrypt and decrypt data. That is, the attacker did not have an advantage over the victim as the exact same key material was available to the attacker and the victim (embedded in the malware). However, early setbacks did not deter ransomware authors. Instead they evolved and turned their attention to asymmetric cryptography. In an asymmetric encryption setting the adversary encrypts the victim's data under a public key, but knowledge of this key does not allow the victim to regain access to the data. Instead, the adversary holds on to the private key and offers the private key to the victim in exchange for the ransom.

Modern ransomware borrows techniques from well established benign cryptography suites such as OpenPGP [23] or S/MIME [39] and employs so-called *hybrid cryptosystems*. In a hybrid cryptosystem, the sender chooses a random symmetric key for each message (e.g., for each file that needs to be encrypted) and encrypts each message (or file) under this key. This one time symmetric key is commonly referred to as a *session key*. Subsequently, a hybrid cryptosystem will encrypt the symmetric message-specific key with the (asymmetric) public key of the recipient. Thus, the performance hungry asymmetric cryptographic operations are only required to encrypt the small symmetric key regardless of the size of the encrypted content. For example, AES, arguably the most popular contemporary block cipher, supports key sizes of 128, 192, and 256 bits which can be trivially encrypted with RSA. The encrypted symmetric key is then combined with the encrypted content and transmitted to the receiver. To decrypt the data, the receiver first uses her private key to decrypt the encrypted symmetric key. With the symmetric key in hand, the recipient can then simply decrypt the cipher text of the data into its original plain text. In a ransomware attack, the attacker generates the asymmetric key pair on his command and control infrastructure. On the victim's machine, the malware generates (and more importantly uses) a unique symmetric session key for each file that is encrypted. The session key is encrypted with the attacker's public key and stored together with the encrypted

file contents. The attacker then offers to sell the private key for the stated ransom.

2.2 Hybrid Cryptography

More formally, an asymmetric encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms where $\mathcal{K}$ is a key generation algorithm, $\mathcal{E}$ is the encryption algorithm, and $\mathcal{D}$ is a decryption algorithm. An asymmetric key pair consisting of public key pk and private key sk is generated as

$$(pk, sk) \xleftarrow{\$} \mathcal{K} \tag{1}$$

A plain text message M is encrypted into a cipher text message $\mathcal{C}$ by an encryption algorithm $\mathcal{E}$ under public key pk as

$$\mathcal{C} = \mathcal{E}_{pk}(M) \tag{2}$$

while adhering to the above stated limitation that $|M| < |pk|$[1]. Of course, to be considered correct, an asymmetric encryption scheme must also correctly decrypt encrypted messages and thus satisfy

$$\mathcal{D}_{sk}(\mathcal{E}_{pk}(M)) = M \tag{3}$$

As explained above, a hybrid cryptosystem combines an asymmetric encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and a symmetric encryption system $\mathcal{SE} = (\dot{\mathcal{K}}, \dot{\mathcal{E}}, \dot{\mathcal{D}})$, such that for each message M a new symmetric encryption key $K \xleftarrow{\$} \dot{\mathcal{K}}$ is generated at random and

$$\mathcal{C}' = \mathcal{E}_{pk}(K) || \dot{\mathcal{E}}_K(M) \tag{4}$$

As Equation 4 illustrates, the symmetric key K is protected (encrypted) with the public key pk. In a ransomware setting the messages encrypted under the hybrid cryptosystem are files on a victim's computer. Thus, if the attacker generates (pk, sk) according to Equation 1 on his command and control infrastructure and only communicates pk to the malware infecting the victim, the strength of the resulting ransomware racket is equal to the security of the hybrid cryptosystem. It is based on this observation that a-posteriori rescue attempts of user files held ransom is challenging at best. Therefore, instead of simply detecting that a victim computer is infected with ransomware, we provide a protection mechanism that sidesteps the challenges of the strong cryptographic primitives employed by modern ransomware samples.

To this end, we leverage the following insights. On the victim's machine we cannot observe the asymmetric key generation algorithm $\mathcal{K}$. However, frequently we have access to the symmetric key generation algorithm $\dot{\mathcal{K}}$ and, more importantly, we can observe the use of the session key during $\dot{\mathcal{E}}_K(M)$ which must be executed to form $\mathcal{C}'$ in the algorithm implementing Equation 4. These observations allow us to capture the symmetric encryption keys that are used to encrypt the files and thus reverse the damage done by the ransomware by trimming off $\mathcal{E}(K)$ from the ciphertext and applying $\dot{\mathcal{D}}_K$ to the remainder.

While the above discussion seems rather theoretic, modern ransomware families leverage exactly such hybrid cryptosystems. Strong cryptography must persist through consistent attacks [42]. Thus, many operating system distributions and platforms contain battle tested implementations of cryptographic algorithms. On Windows, one such implementation is Microsoft's CryptoAPI. The CryptoAPI is a secure interface for cryptographic functionality that is guaranteed to be present on every Windows installation and thus makes it trivial for ransomware authors to leverage the existing functionality.

Appendix A, provides ransomware pseudocode using Microsoft's CryptoAPI. Note that both CryptoWall and CryptoLocker, two of the most successful ransomware families, use exactly the same APIs for their nefarious purposes.

[1]The inequality for the RSA asymmetric encryption scheme is slightly stricter: $|M| < |pk| - 11$ bytes.

2.3 Threat model

The threat model and assumptions for our proposed system are captured in this section. A detailed discussion of the assumptions and why we deem them realistic is deferred to the discussion section in §6. Our threat model is based on modern economically successful ransomware families. Thus, the threat model considers an attacker who installs malware on victim computers through an established malware distribution channel. Furthermore, the operating system in our threat model is trusted and updated. Thus, we assume that the malware does not elevate privileges, as this would defeat any existing in-host protection mechanisms (e.g., anti-malware solutions) too. While we assume the ransomware only executes with user-level privileges, most contemporary malware is packed. Therefore, our threat model assumes that the malware is packed with common-off-the-shelf packers. More precisely, the threat model only considers packers that unpack the whole binary at once and not those that apply advanced strategies such as incremental unpacking and repacking or emulation-based packers such as Themida [14]. We acknowledge that more sophisticated packers and obfuscation techniques can defeat our proposed system. While such techniques have been known in scientific literature for years (e.g.,[43]), these techniques have not gained traction in the malware community at large. Whatever the reason for the slow uptake, we argue that our proposed approach significantly raises the bar for malware authors to evade detection (i.e., attackers would have to overcome these reasons). Finally, we assume the user can create an uncompromised asymmetric keypair to activate our system and does so before being infected with ransomware.

3. OVERVIEW

PayBreak consists of three different components which are combined to form a cohesive system that is able to reverse file encryption performed by hybrid cryptosystem ransomware. In this section we provide an overview of these components and their roles in the system. Figure 1 shows a working scenario of PayBreak. Upon installation, the user configures PayBreak with the public key (pk_u) of an asymmetric key pair (pk_u, sk_u), while the private key (sk_u) is secured off site. The system continuously stores encrypted session keys used on the machine in a secure key vault. In the unfortunate event that the user's machine is infected with ransomware, the system's key vault is accessed with the private key, sk_u. The data stored in the key vault is then used to decrypt ransomed files granting the user a pay break.

The system leverages the fact that in a hybrid cryptosystem the session key must be used during the symmetric encryption. In a practical ransomware attack this encryption must happen on the user's machine. Because of this char-

acteristic, we can sidestep the challenges imposed by the strong cryptography employed by modern ransomware.

Crypto Function Hooking.
Ransomware authors require secure cryptography for long term success. The reason is that, historically, developing their own implementations was met with cryptographic defeat. Thus, today's malware authors can choose to either dynamically link against (system-provided) cryptographic libraries, or statically link external libraries into their code. PayBreak supports both types of linking behaviors and identifies procedures in dynamically linked libraries by their name and address, whereas statically linked procedures are identified based on fuzzy byte signatures. *Hooks* are then created at the location of these procedures. The hook redirects control from these cryptographic procedures and exports session keys, and any parameters for the symmetric encryption scheme. Once data is exported, the system then returns control back to the original cryptographic procedure, and flow continues as normal. The details of this component are discussed in §4.1.

Key Vault.
The key material and algorithm details (recovered from hooked procedures as explained above) to recover symmetrically encrypted data are stored in a safe and securely encrypted key vault. Due to the threat of ransomware targeting the key vault, our implementation stores the harvested key material into an append-only file protected with Administrator privileges. This integrity mechanism has shown to be sufficient in our evaluation. However, we discuss further key vault integrity improvements in §6. The contents that enter the key vault are securely encrypted with the user's public key. By encrypting the data prior to storing it, we ensure that the key vault is secure for the user. The details of this component are discussed in § 4.2.

File Recovery.
In the unfortunate, but rare situation where a user gets infected with ransomware and her files are held ransom, the key vault is accessed with the user's private key, sk_u. PayBreak is used to access the key material and algorithm details used to encrypt the files being held ransom. The algorithm details are used to configure the appropriate symmetric encryption scheme and the key material is used with this configuration to attempt recovery. Because, ransomware typically stores meta data, such as the original file length, the date of encryption, and encrypted key data, at the beginning of encrypted files the actual encrypted file data is often offset by this meta data. Prior to decryption with vaulted symmetric keys, PayBreak determines the correct offset into the encrypted file. The details of this component are discussed in § 4.3.

4. IMPLEMENTATION

We implemented our prototype system, PayBreak, to target hybrid cryptosystem ransomware on the Windows 7 operating system. Our implementation is configured to hook encryption performed by Microsoft's Crypto APIs and the Crypto++ library. The implementation additionally uses the strong cryptography in Microsoft's CryptoAPI to securely store session keys employed by ransomware. In the unfortunate event of a ransomware infection, these keys may be retrieved by the user of PayBreak to be used for the decryption of files held for ransom.

4.1 Crypto Function Hooking

Hooking is a scheme that is used to modify application behavior by augmenting original functions with arbitrary new functionality. In Windows, functions can be hooked by various means, ranging from modifying a processes' Import Address Table, to injecting DLLs [19]. Our prototype employs Microsoft Research's Detours library for hooking. Detours hooks a function by first saving a minimum of 5 bytes (the size of an unconditional JMP instruction in x86 assembly) from the beginning of the original function's memory address into a new hook function. This specific amount saved might extend past 5 bytes due to variable length instructions in the x86 architecture. The hook function also contains the new functionality that is added. For PayBreak this is the code that exports the session keys and algorithm parameters to the key vault. At the end of the newly created hook function, Detours creates an unconditional jump instruction that transfers control to the original function skipping over the bytes already saved into the hook function. To activate the hook and redirect control from the original function to the hook function, an unconditional jump instruction to the hook function overwrites the first five bytes in the original function. This completes the hook, and any calls made to the original function, will now be redirected to the hook function. Our system employs this scheme for hooking and injects itself into every new process launched on a Windows 7 machine.

Ransomware authors include cryptography into their malware by dynamically linking against system provided cryptographic libraries, or statically linking external libraries. These two linking behaviors pose different challenges to the hooking portion of the system.

Hooking in dynamically linked libraries.
Windows has included feature rich cryptographic libraries as part of their platform for decades. This ubiquitous presence makes it easy for malware to dynamically link to the cryptographic libraries found on Windows machines. Microsoft's CryptoAPI hides sensitive information such as keys and their locations in memory by only allowing operation through a set of subroutines that have special access to it. The security, platform consistency, and API completeness of the CryptoAPI makes it a common choice for local file encryption by ransomware authors. Microsoft's CNG library, is the long term replacement for the classic CryptoAPI (both are included in Windows 7), but operates much the same way, and is seamlessly handled by PayBreak too.

Due to the abstract and opaque design of the CryptoAPI, usage and exportation of the session key can only be accomplished through specific CryptoAPI procedures. All encryption through the CryptoAPI must be performed via the `CryptEncrypt` function, or be exported (i.e., for external use) via the `CryptExport` function. CryptoAPI based ransomware uses the `CryptEncrypt` function of the CryptoAPI to perform local encryption of files. Because, the CryptoAPI is dynamically linked, adding a hook is completely independent of the calling process, and malware obfuscation does not impact this capability. Through a hook configured in `CryptEncrypt`, PayBreak securely exports the session key

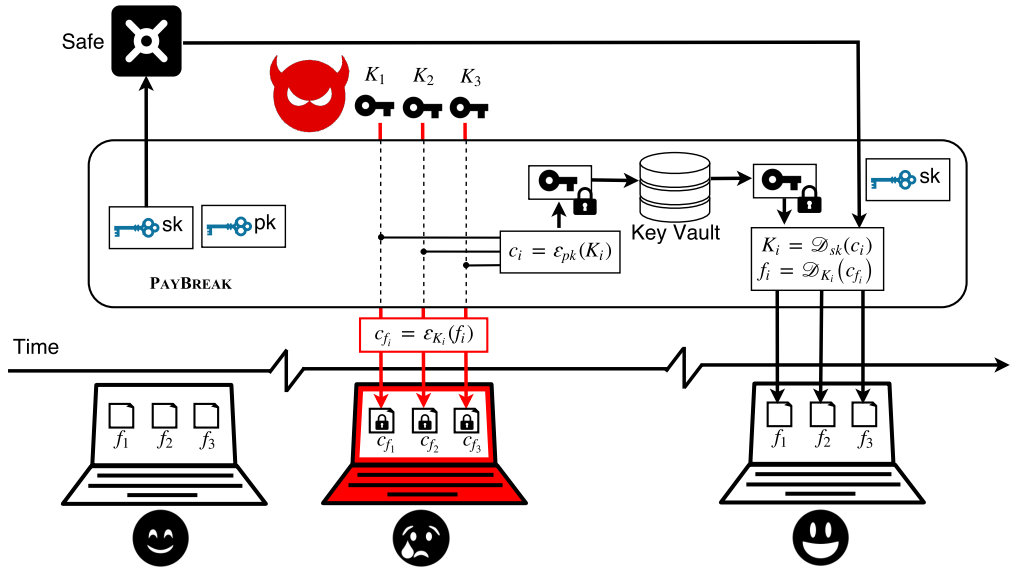

Figure 1: Overview of PayBreak.

using the `CryptExport` API function. While the hook in `CryptEncrypt` successfully exports the session key, details such as the cipher mode, and the initialization vector are not included. To obtain these parameters and then recreate the appropriate configuration, our system hooks the `CryptAcquireContext`, and `CryptSetKeyParam` functions. The hook into `CryptAcquireContext` provides PayBreak with knowledge of the algorithm that was used for encryption, including the default parameters. Changes to these parameters are performed with the `CryptSetKeyParam` function, as such this API function is hooked as well.

Alternatively to encrypting with the CryptoAPI, a user may wish to generate a cryptographically secure random number with the API instead. The random number can then be used to derive a session key for another encryption function. In Window's case, the supported API to generate a random number is `CryptGenRandom`, and many cryptographic libraries (OpenSSL, NaCl, LibTomCrypt, and more) leverage this API for their cryptographically secure pseudorandom number generator (CSPRNG). By dynamically hooking, and recording this system function, PayBreak stores the base material used to generate many session keys used by ransomware linking these libraries either dynamically, or statically.

Hooking in statically linked libraries.

Ransomware that statically links a cryptographic library forces PayBreak to follow a slightly different approach. Statically linked libraries are embedded into the executable code of the application, and thus are affected by obfuscation. Thus, PayBreak identifies cryptographic procedures at runtime in a process' memory, and subsequently hooks them. To this end, our system uses 32-byte fuzzy function signatures to identify statically linked library functions. This approach is similar to IDA's Fast Library Identification and Recognition Technology (FLIRT) [7]. A signature consists of the first 32 bytes of a known procedure and a procedure is identified when a threshold percentage of these 32 bytes is identified contiguously in memory. Because, malware is typically packed, PayBreak scans executable memory of all executed processes for function signatures. Our prototype implementation performs a scan after the first `NtReadFile` system call in each process. The rationale is, that in order to encrypt user data, the malware must read the data first. When a signature is identified, a hook is placed at its address through the use of Detours. This hook securely exports the session key, and encryption algorithm details. While our current prototype is effective against contemporary ransomware, advanced packing and obfuscation can thwart the system (§6). An avenue to strengthen the detection of cryptographic code could leverage the semantic detection of cryptographic functionality such as the analysis presented in [35].

Our prototype implementation is outfit with signatures for the Crypto++ statically linked library. These signatures consist of the first 32 bytes of Crypto++'s `SetKey`, `CipherModeBase`, and `SymmetricCipherBase` class methods. The export of Crypto++'s session key, and algorithm details is done with the CryptoAPI's `CryptExport` API function. We evaluate and discuss the robustness of our signatures in §5.2.

4.2 Key Vault

The system's key vault asserts that symmetric keys used by Microsoft's CryptoAPI and Crypto++ are securely stored only to be accessed by the ransom victim when necessary. The details about the symmetric encryption schemes are stored as well. Ironically, PayBreak's key vault system is designed similarly to the hybrid cryptosystems that ransomware deploys. Session keys are encrypted and exported using the user's public key (pk_u) generated during installation of the system. Our implementation uses 2048-bit RSA keys for this step. The large key size of 2048 bits guarantees secure encryption of data in size less than or equal to the key size — more than enough for typical 256-bit symmetric keys.

As explained in the previous section, a call to `CryptEncrypt` is augmented with the behavior of the `CryptExport` function. The auxiliary `CryptExport` call takes as arguments the handle to a session key that is passed to the `CryptEn-`

`crypt` function, as well as our system's exchange key (i.e., the user's pk_u) to securely export the session key. Keys being used by `CryptEncrypt` also contain algorithm (i.e., AES, 3DES, RC4, etc.) information, and as such this information is exported as well. Additionally, in order to reconstruct the symmetric encryption configuration used by a ransomware infection, algorithm parameters such as the initialization vector (IV), and the block cipher mode used must be saved. This information is extracted from hooks that perform recording of the parameters passed to the `CryptAcquireContext` and `CryptSetKeyParam` functions. Akin to the Cryptographic Message Syntax[2], these parameters are concatenated to the session key material in cleartext, as their disclosure does not affect the security of modern crypto systems. This concatenated 'blob' is appended to PayBreak's key vault. Additionally, as described in §4.1, our prototype implementation stores the cryptographic key material (simple byte arrays) passed to Crypto++'s functions into the key vault. The system also stores the outputted random bytes from `CryptGenRandom` function calls. These bytes can be used on a reverse engineered ransomware family to recreate session keys used to encrypt files.

As a safety precaution to prevent the vault itself from being encrypted by ransomware, the vault is configured to be append-only and all other access is only allowed to the Windows Administrator group. In the unfortunate event that the key vault requires access, the private key (sk_u) set up during installation of PayBreak is used to decrypt the key material stored. This yields access to the individual session keys, and encryption scheme parameters.

4.3 File Recovery

The last component of PayBreak is the actual recovery of the files encrypted during a ransomware infection. File recovery works in three phases. First, the key vault is accessed using the stowed away private key. Second, the data in the vault is parsed into the symmetric keys and the corresponding encryption scheme parameters such as, block cipher mode, and initialization vector. Finally, the retrieved session keys are then used to decrypt the victim's files.

Each file encrypted by ransomware is typically concatenated with meta data, such as the ransomware version, and original size of the encrypted file. Because of this meta data the actual encrypted file data is often offset in files held for ransom. Without knowledge of each ransomware family's individual meta data structure, our system is forced to test decryption at every possible offset in the files held for ransom. Our system leverages dynamic programming to lower the effort required for subsequent file decryption, once a successful offset is found, future file decryption attempts will be attempted at the previously successful offset.

PayBreak iteratively attempts decryption of a file with each escrowed key and each offset, until a decryption state is reached that is not identified as 'data' by libmagic[3]. Once a decryption state is identified as a common office document file type, such as a Microsoft Word Document, JPEG image, or PDF file, the state is saved as the actual decrypted file. Of course, if the resulting file is incorrectly flagged as decrypted, the user can instruct the system to continue its search until the right key and offset is identified. While this possibility exists, we did not encounter it during our evaluation. Furthermore, albeit this unoptimized brute-force approach can be improved, it is successful in recovering encrypted user files as detailed in § 5.2.

[2]CMS, http://www.ietf.org/rfc/rfc5652.txt

[3]Fine Free File Command, http://www.darwinsys.com/file/

5. EVALUATION

As illustrated in §4, we implemented PayBreak for the Windows 7 operating system. Based on this prototype implementation we performed the evaluation described in this section. The goal of the evaluation was to answer a set of research questions:

RQ1 Can PayBreak protect users from the threat of real-world ransomware (i.e., can PayBreak restore files encrypted by the malware)?

RQ2 Are malware family specific modifications necessary to revert the encryption employed by different ransomware families?

RQ3 What are the performance impacts caused by running PayBreak as a *proactive* online protection mechanism?

These questions are geared toward answering practicality concerns of the proposed technique. RQ1 addresses whether the technique is effective. Clearly, a system that cannot answer RQ1 in the affirmative, would be of limited help in the fight against ransomware. RQ2 explores the versatility of the proposed system. Here a versatile approach is preferable over a technique that requires continuous refinement to address the challenges of previously unknown ransomware families. Finally, RQ3 addresses a practical deployment question. Similar to popular anti-virus solutions, we designed PayBreak as an online protection mechanism. Thus, high performance impacts on common use-cases and workloads would pose significant hurdles to the adoption of such a mechanism in the field.

5.1 Dataset

To test the functionality and effectiveness of PayBreak, we needed access to actively encrypting ransomware samples. To collect these samples, we developed the Real-time Automation to Discover, Detect and Alert of Ransomware (RADDAR) system. This project will be made open source to help further research in ransomware. RADDAR crawls various locations for malware samples. More precisely, we obtained samples from VirusTotal Intelligence[4] which provides advanced search features and download functionality for malware samples. We searched for newly submitted samples (i.e. submitted within a week of analysis) that were also flagged by at least two anti-virus vendors, and whose labels contained known ransomware defined in [10]. Beyond these popular ransomware family names, we also download samples for the generic search terms: ransom, crypt, or lock. In addition to VirusTotal, we crawl various malware repositories including Malc0de[5], and VXVault[6] similar to the efforts presented in [33].

Once RADDAR discovers a malware sample, it detects whether or not the malware sample is crypto-based ransomware and whether it is actively performing its malicious duties. To this end, we leverage the Cuckoo Sandbox[7] dynamic analysis framework where each sample is run for 20

[4]VirusTotal, https://www.virustotal.com

[5]Malc0de, http://malc0de.com/rss

[6]VXVault, http://vxvault.siri-urz.net/URL_List.php

[7]Cuckoo Sandbox, https://cuckoosandbox.org/

minutes. We use Cuckoo to analyze and output a behavior report of each sample by executing it in a monitored Windows 7 virtual machine (VM) running in KVM[8]. Furthermore, in addition to the default files found on a clean Windows 7 installation, we placed typically ransomed file types (PDFs, images, source code, and Word documents) across various directories on the machine. Finally, we added PayBreak to the VM which allowed us to perform the measurements presented in this evaluation. We took a snapshot of the file system, and refer to these files found on the system prior to an infection as "honey files."

After a malware sample is analyzed by Cuckoo, RADDAR performs an analysis on the Cuckoo results to generate a report containing a variety of metrics including if the sample is active, and if PayBreak extracted the keys used during encryption. We consider a ransomware sample active if it: (1) overwrites, or deletes and recreates at least one honey file, and (2) the new file is identified as `data` by libmagic. Note that libmagic successfully identifies the true content of the honey files in their original state, thus if the type changes to data the sample must have modified it.

To determine the ransomware's family, we perform majority voting of AV labels (i.e., following the same approach as Kharraz, et al. [34]).

We let RADDAR run for 4 months to collect and generate reports for 1,691 malware samples, 713 of which match ransomware labels used by AV companies. Figure 2 presents a detailed breakdown of this analysis.

Consistent with previous malware research, many of the samples in our dataset did not show any malicious functionality (i.e., they were inactive) for a variety of reasons. Thus, we performed the following two step analysis. First, identify active samples, and second try to infer the reason why inactive samples did not show any malicious behavior. As previously discussed in §2, ransomware that uses hybrid encryption securely must retrieve the public key pk from the command and control infrastructure (C&C). Therefore, if the malware does not produce any network traffic, it cannot apply hybrid encryption securely. We classify a sample as "No Network" if all observed TCP and UDP traffic exclusively targets Windows affiliated domains, such as ones for time keeping and updating.

Furthermore, we classify a sample as "Disabled C&C" if either (1) all DNS queries (beyond those for the Microsoft domains) return negative, or (2) all HTTP requests result in a 404 status code. Of the inactive samples analyzed, those reported as having a disabled C&C are all due to DNS queries returning negative. However, we did not introspect into HTTPS protected network traffic generated by the malware.

Finally, even if the C&C is operational and reachable, environment sensitive malware will refrain from executing if it detects that it is run in a sandboxed environment. "Environment" indicates that the malware may be analyzing its environment to detect if it is running in a virtual environment such as KVM or VirtualBox. A sample is labelled "Environment" if Cuckoo's built-in detection identifies it as such.

The reason for the inactivity of the remaining samples could not be determined. Previous experience suggests, this could be due to advanced environment fingerprinting, dependency on user activity, or the use of logic (time) bombs that delay execution beyond our 20 minute evaluation threshold. In the end, we evaluated our system against 20 active ransomware families, the largest study of ransomware we are aware of.

[8]KVM, http://www.linux-kvm.org/page/Main_Page

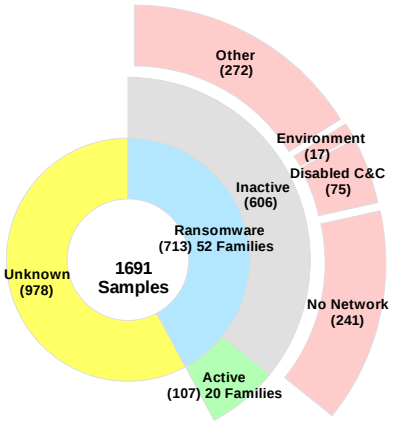

Figure 2: Summary of analysis results. Sample count for each category is in parentheses.

5.2 PayBreak Effectiveness

In this section we answer **RQ1**, can PayBreak revert the encryption performed by real-world ransomware? And **RQ2**, are malware family specific modifications necessary to revert the encryption employed by different ransomware families?

PayBreak is able to recover ransomed files from all ransomware families with known encryption signatures. Our results confirm that we are able to successfully hook into the encryption functions of real-world ransomware samples and extract session keys, and all materials necessary to be used for file recovery.

More specifically, PayBreak defeated 12 of the 20 active ransomware families, 9 of which, according to our knowledge, have not been previously defeated. We define a ransomware family as defeated if there exists a method or technique for which the ransomed files can be fully recovered. PayBreak successfully recovers files encrypted by CryptoWall, and Locky, two of the three economically most successful ransomware families of 2016 [2]. CryptoWall alone has netted more than $325 million in revenue [3].

A summary of the active ransomware families is shown in Table 1. The number of active samples for a given family is specified in the Samples column. For ransomware samples that have previously been defeated, a corresponding reference is included in the column. We do not consider leaked cryptographic keys (e.g., obtained from a confiscated command and control server) as defeated due to this being specific to a ransomware campaign, and does not imply a weak implementation of the ransomware family. PayBreak has demonstrated the ability to defeat ransomware using multiple cryptographic libraries including Microsoft CryptoAPI and Crypto++. The cryptographic algorithms used for encryption were extracted at runtime by PayBreak for samples that PayBreak defeats and are listed in the Algorithm

[9]Razy cryptographic information identified by Jakub Kroustek, malware analyst at Avast.

	Families	# Samples	Previously defeated	Defeated by PayBreak	Defeated	Library	Algorithm
	Almalocker	1	✓ [1]	✓	✓	CryptoAPI	RSA+AES-128-CBC
	Cerber	14	✓ [4]	✓	✓	CryptoAPI	RSA+RC4-256
	Chimera	1	✗	✓	✓	CryptoAPI	RSA+AES-256-ECB
	CryptoFortress	2	✗	✓	✓	CryptoAPI	RSA+AES-256-ECB
	CryptoLocker	33	✗	✓	✓	CryptoAPI	RSA+AES-256-CBC
	CryptoWall	7	✗	✓	✓	CryptoAPI	RSA+AES-256-CBC
	CrypWall	4	✗	✓	✓	CryptoAPI	RSA+AES-256-CBC
	GPcode	2	✓ [38]	✓	✓	CryptoAPI	RSA+AES-256-ECB
	Locky	7	✗	✓	✓	CryptoAPI	RSA+AES-128-CTR
	SamSam	4	✗	✓	✓	CryptoAPI	RSA+AES-128-CBC
	Thor Locky	1	✗	✓	✓	CryptoAPI	RSA+AES-128-CTR
	Tox	9	✗	✓	✓	Crypto++	RSA+3DES-128-CBC
	DXXD	2	✓ [5]	✗	✓	Unknown	XOR with Constant Key [5]
	MarsJokes	1	✓ [12]	✗	✓	Unknown	ECC+AES-256-ECB [12]
	PokemonGo	1	✓ [9]	✗	✓	.NET Crypto	AES with Constant Key [9]
	Troldesh	5	✓ [13]	✗	✓	Unknown	RSA+AES-256-CBC [13]
	VirLock	4	✓ [15]	✗	✓	Unknown	XOR with Constant Key [15]
	Androm	2	✗	✗	✗	Unknown	RSA+AES-256-CBC [11]
	Razy	3	✗	✗	✗	.NET Crypto	AES-128[9]
	TeslaCrypt	4	✗	✗	✗	Unknown	ECC+AES-256-CBC [16]
Total		20	107	8	12	17	

Table 1: Active Ransomware Samples

column. For undefeated samples, the column contains information reported by other researchers, gathered to the best of our ability with a corresponding reference. Additionally, the Defeated column identifies all families that have been defeated, either by a previous technique or PayBreak.

Of the 20 active families in our dataset, PayBreak does not defeat eight. Three of these, DXXD, PokemonGo, and VirLock, are previously defeated and use trivial constant keys for encryption, i.e., they are not using hybrid cryptography. Two other families, MarsJokes and Troldesh, have also been defeated previously. These families, contrary to popular reason, rolled their own pseudorandom number generators instead of using the battle tested `CryptGenRandom` API. Their naïve implementations lead to their defeat by researchers [12, 13]. The remaining undefeated families, Androm, Razy, and TeslaCrypt use a cryptographic library that our prototype implementation was not set up to hook. PayBreak can be expanded to defeat all eight of the remaining families by hooking their respective statically linked encryption functions, and either exporting their constant keys (in the case of the trivial families), or their session keys into the key vault. We next discuss the robustness of the used signatures.

Signature Robustness.

To identify statically linked crypto libraries, PayBreak relies on signatures. Thus, one obvious concern is the robustness of these signatures against obfuscation. Identically to all practical online anti-malware solutions, sufficient levels of obfuscation and deception can evade the protections afforded by PayBreak. Note, however, that packers that unpack the entire binary at runtime do not pose a problem for PayBreak. To evaluate the robustness of our signatures, we assess them against the syntactic changes introduced by different compilers and optimization levels as these characteristics can be easily changed by the attacker. To this end, we compiled 12 programs that use the Crypto++ encryption library with different compilers and optimization settings. More precisely, our sample programs statically link Crypto++ versions 5.6.3, 5.6.2, and 5.6.1, spanning half a decade of development of this popular cryptographic library for Windows. Furthermore, we compiled our programs with the `tdm-gcc`[10] and the `mingw32-gcc`[11] compilers, each with disabled optimization, and maximum optimization levels. To identify all 12 variants of the encryption functions, we had to develop two signatures. The reason is that the differences between artifacts is large when using different compilers, but smaller for different optimization levels. In essence, we had to create one signature per compiler, and each signature was robust across all tested optimization levels and library versions.

File Recovery.

PayBreak is able to fully recover encrypted files from twelve families. Due to the large amount of samples we're working with, our RADDAR system executes each sample for 20 minutes. However, to evaluate the performance, and ability to recover entire file systems we executed three ransomware families for four hours each in a resettable test environment. To develop this test environment, first, we spread across the standardized document corpus, Govdocs1 threads [28], randomly across the entire file system of a virtual machine. The document corpus contained 9,876 files primarily of common office types, such as `.xls`, `.docx`, and `.pdf`. For each of the files we recorded their original SHA1 file hash. By comparing these file hashes we can determine if our recovered file is the original file. We then executed a ransomware family and let it run for four hours without intervention. After this infection, we extracted all files on the system to a safe environment. PayBreak attempted re-

[10]TDM-GCC, http://tdm-gcc.tdragon.net/
[11]MinGW, http://www.mingw.org/

covery on these files using the key vault extracted off of the system. We then reset the virtual machine, and repeated this process for each family.

After ransomware encryption of the file system we executed PayBreak decryption. Our system was able to recover 100% of the original encrypted files from each of the attacks. Comparing to the previously generated original file hashes is not essential for the recovery, and using the file hashes previously generated only serves as a confirmation of successful file recovery. The Locky sample encrypted 9,821 files which were recovered in 360m40s. The Cryptowall sample encrypted 204 files from our document corpus, and files were recovered in 86s. The Alma Locker sample encrypted 271 files from our document corpus, and the affected files were recovered in 26s. The Cryptowall, and Alma Locker samples encrypted a low amount of files probably due to malware software instability, i.e. they crashed during their execution; however, nonetheless these tests prove that PayBreak is able to *fully recover all files* from a ransomware attack in a short amount of time, i.e. on the scale of a few hours for a full file system.

5.3 Performance Impacts

In this section we answer **RQ3** which assesses performance impacts caused by PayBreak. For this question we are interested in two characteristics. (1) what slowdown does PayBreak introduce for a single call to the encryption API (i.e., micro benchmark), and (2) with what frequency are calls to the encryption API made during regular office workloads (i.e., macro benchmark). We assessed performance on a consumer grade laptop, running a Windows 7 32-bit virtual machine, with 2GB of RAM, and 2 CPU cores at 2.20GHz.

Micro benchmark.

In order to measure the overhead of our system on CryptoAPI functions hooked with Detours, we performed a micro benchmark of 10 million invocations of the `CryptEncrypt` API on a 1KB file. We found that it takes 4.02s without our hooks in place. With PayBreak enabled, exporting session keys and encryption scheme information to the key vault, the tight encryption loop took 1,242s. Thus, on average one call to the `CryptEncrypt` API takes 124μs (i.e., a slowdown of a factor 310x). However, the majority of the performance impact results from the I/O operations that write to the key vault. Omitting the disk I/O from the measurement reduces the slowdown to a factor of 1.5x. Thus, a simple performance optimization of PayBreak could be to perform the write operation to the key vault in a dedicated I/O thread. We will now show that realistic workloads suffer from significantly less performance impact than the synthetic worst-case benchmark discussed above.

Macro benchmark.

While relative performance impact on a single invocation of the cryptographic APIs is significant, such operations are extremely rare in regular office workloads. For our macro benchmark we used common Windows software on a virtual machine provisioned with PayBreak. The Windows software we executed included: 7zip, AVG, Dropbox, Firefox, Gimp 2, Google Chrome, Google Drive, Internet Explorer (IE), iTunes, KeePass 2, LibreOffice, Microsoft Excel, Microsoft Powerpoint, Microsoft Word, Pidgin, Putty, RealVNC, Skype, SumatraPDF, WinSCP, WinZip.

Due to space limitation, we are unable to thoroughly discuss our testing procedure for each application we tested. However, we provide insight into the analysis of five applications. We found no significant slowdown in any of the applications, and on average less than 100 cryptographic API calls during regular application usage. The number in parentheses following each applications name is the number of CryptoAPI calls we recorded stemming from the application during its testing.

KeePass 2 (28). We created a new password database and randomly generated 3 passwords using the application. We deleted that database, and created a new empty database. We then imported an old database. We noticed no slowdown in any of these operations, and the application worked completely normally. KeePass 2 appears to be a diverse user of the CryptoAPI, as we observed six different functions from the CryptoAPI being called.

Dropbox (127). We logged into a Dropbox account using the program. We then synchronized 3 files from that account previously placed there onto the local machine. We then synchronized 5 files from the local machine onto the cloud by dragging the files into the Dropbox folder. We noticed no slowdown during the synchronizations, and no program instability. Most CryptoAPI calls made by Dropbox during our testing were to `CryptGenRandom`.

Putty (2). We connected to a remote SSH server. Upon connect, we executed several commands. We then disconnected from the server. This application is a sparse user of the CryptoAPI. We observed no slowdown, or program instability.

Skype (19,418). We created a Skype account. We then added and messaged 2 contacts. We then called 1 of those contacts. Skype is a very heavy user of the CryptoAPI, more than any other program that we observed. However, even with such heavy usage, far above any other program, we noticed no slowdown, or program instability.

Internet Explorer (3,328). We utilized the AutoIt[12] program to automate IE to visit the Alexa Top 100 most popular websites on their HTTPS front pages. We stayed on each page for 5 seconds to allow page loading. On average, we found 33 calls to the CryptoAPI per webpage (including all resources). Thus, even at an unoptimized slowdown of 124μs per cryptographic operation, this resulted in 4.1ms overhead for page loads and is clearly below the human perception threshold [37].

6. DISCUSSION AND LIMITATIONS

In this section we discuss challenges, open problems, and limitations that exist despite or due to our system. A trivial and seemingly effective defense against ransomware is a reliable backup regime. With such a system in place, users have little to fear from a ransomware attack. All it takes to recover is wiping and reinstalling the infected machine and restore the data from backup. Although simple, it is fair to assume that users who fell victim to ransomware and payed the ransom in the past, did not have this simple mechanism in place. Unfortunately, comprehensive use of backups by all users seems unrealistic.

Furthermore, some recent ransomware families (e.g., RansomWeb or CryptoWall) are reported to encrypt files immediately upon infection and providing access to the data for a

[12]AutoIt, https://www.autoitscript.com/site/autoit/

limited amount of time (e.g., a few months) by transparently decrypting the data when accessed [32, 26]. Once this initial period expires, the malware destroys the key that is necessary for the decryption and asks for ransom. At this point, all backups taken since the infection (i.e., months worth) contain only encrypted data and are thus useless to recover from the infection.

At its core, PayBreak is a key escrow system. Government proposed and mandated key escrow systems have consistently been met with well founded criticism from the research community and privacy advocates. We absolutely sympathize with this opinion and strongly oppose government mandated key escrow systems. However, there is a fundamental difference between such government mandated proposals and PayBreak. In PayBreak there exists *exactly one* entity who has access to the keys kept in the key vault — the legitimate user herself. That is, besides the user herself, there are no trusted third parties.

Our prototype implementation of PayBreak defeats ransomware using several dynamically or statically linked libraries. Ransomware authors may be tempted to roll their own cryptographic libraries in an effort to evade PayBreak; however, this is often a recipe for easy cryptographic defeat (for example, simple encryption used by botmasters made their C&C protocol easy to reverse engineer [40]). As such, leveraging secure third party libraries is appealing to ransomware authors. However, secure crypto libraries are scarce, and only a limited pool to choose from exists. Nonetheless, for our system, creating signatures for third party libraries, or custom libraries is not any different. Our experience working with three different libraries suggests that adding support for further libraries can be achieved easily, and quickly. For example, we developed the signatures required to detect Crypto++ within one day. Additionally, our prototype hooks the Windows standard CSPRNG function, `CryptGenRandom`. By dynamically hooking, and recording this system function PayBreak stores the base material for session keys used by any ransomware leveraging many of these alternative cryptographic libraries. Regardless what code is used, malware analysts only have to identify the encryption implementation once, and add it to PayBreak to add support. Identification of cryptographic code does not have to be a manual effort. Instead, the process may be automated through a variety of means (e.g., [31]). Once cryptographic code is identified, a wealth of work (e.g., [7, 17, 20, 27]) exists that aides in identifying similar code. To sidestep the use of symmetric keys altogether, ransomware authors might be tempted to encrypt data with exclusively asymmetric primitives. While such a strategy is possible, the high resource requirement and uncharacteristically frequent use of asymmetric encryption could be addressed with heuristics that monitor for these aspects. Despite PayBreak's demonstrated effectiveness against contemporary cryptographic ransomware, a practical deployment would have to address several issues that are out of scope for this paper. These issues include, for example, teaching users to keep private keys secure, or implement a secure rotation system for the key vault to prevent unlimited growth of the vault.

As evaluated in §5.2 PayBreak is able to recover from a ransomware attack in hours. As described in §4.3, file recovery is independent of the ransomware family by means of exhaustive search. This exhaustive search is an embarrassingly parallel workload and thus can be optimized almost arbitrarily with additional compute resources, such as a cloud deployment. Furthermore, needing to recover from a ransomware infection should be a rare and exceptional situation. We submit that for a regular ransomware victim regaining access to encrypted data is paramount when compared to the speed with which encrypted files can be recovered.

We acknowledge that, identically to all practical online protection systems, obfuscation and evasion can thwart the protections afforded by PayBreak. However, obfuscation is only a concern for malware that statically links against cryptographic libraries. Protection from families that use the system-provided CryptoAPI is unaffected, as PayBreak hooks the implementations of the API functions in the unobfuscated system DLLs. Furthermore, as our evaluation shows, PayBreak is perfectly capable of protecting users from malware that statically links cryptographic libraries, as long as the malware is obfuscated with contemporary packers. In fact, all malware samples used for the evaluation in this work were packed. Moreover, the samples from the Tox family statically link the Crypto++ library. The academic literature (e.g., [43]) and commercial sources (e.g., [14]) have offered advanced obfuscators for years. However, these advanced techniques have not gained significant traction in the malware ecosystem at large. For example, Sun [44] reports that 91% of the 103,392 analyzed samples were packed with simple packers, such as UPX or ASPack, which do not inhibit the protections afforded by PayBreak. Unfortunately, we do not know the reasons that prevent malware authors from making wide-spread use of more advanced obfuscation techniques. PayBreak thus raises the bar for malware authors and forces them to use evasive techniques that they so far have resisted to employ.

Another evasive strategy that malware can implement to evade PayBreak is to detect that PayBreak is running in the victim's computer, and accordingly jump over the inserted hooks. However, there is no reason that PayBreak must install the hooks at the beginning of the targeted cryptographic functions. We could modify PayBreak to insert the hooks are arbitrary points in the function as long as the relevant data structures (e.g., keys and encryption scheme parameters) are still in scope. Similar to the obfuscation scenario, PayBreak cannot provide guarantees against malware that specifically aims to evade PayBreak. But PayBreak can significantly raise the bar for attackers' success.

Finally, ransomware that is aware of PayBreak could try to launch a denial of service attack by either corrupting the public key used to escrow data in the vault or simply fill the vault with nonsensical information. PayBreak can be modified to have a dedicated (privileged i.e., running as `SYSTEM`) process that appends to the vault and therefore protect the integrity of the public key. An attack that fills the vault with garbage before encrypting the victim's files can only hope to increase the time it takes to recover encrypted files. The privileged process mentioned above could detect that such an attack is ongoing and alert the user, or terminate the offending process. If this attack were to occur unnoticed, recall that getting infected with ransomware is a rare circumstance and identifying the correct key and encryption offset is embarrassingly parallel. As such, even with a large vault (a 1TB vault can hold roughly 17 billion entries), recovery would only be delayed, but not prevented.

7. RELATED WORK

Decades ago (in 1996) Young and Yung were the first to introduce the concept of a *cryptovirus* [46, 47]. Their description is a perfect blueprint of today's most effective crypto-based ransomware families that leverage hybrid encryption to blackmail users into paying ransom for regaining access to their data. The only difference between the proposed cryptovirus and modern ransomware is that the Internet made it trivial for malware authors to generate an individual asymmetric key pair for each infected victim and communicate the public key to the malware via a command and control channel. In 2005, the same authors presented a proof-of-concept implementation of their cryptovirus, using the Microsoft Cryptographic API [48]. As possible mitigations against this attack they proposed to restrict access to the cryptographic API, by for example only allowing encryption with keys that come from a trusted certificate. As the authors' call to control access to cryptographic tools was ignored, it comes at no surprise that malware authors leveraged cryptography for their malign purposes.

In 2010, Gazet studied three early families of ransomware, highlighting how this threat was not mature for mass extortion at the time [29]. In 2015, Kharraz et al. [34] presented a longitudinal study of 1,359 ransomware samples collected between 2006 and 2014. While their study included four families that encrypt user data, the majority of the analyzed families were either deleting files or exfiltrating information. Furthermore, the paper presents measurements of I/O characteristics of ransomware samples and states that these characteristics are sufficient for a monitoring mechanism to distinguish ransomware from benign applications. However, the paper does not evaluate such a mechanism.

In follow up work, Kharraz et al. [33] implemented and evaluated the proposed mechanism as an offline malware analysis system and found it to have a 96.3% accuracy in detecting ransomware. However, the monitoring mechanism is inherently reactive. That is, repeated file operations leading to frequent I/O could indicate that ransomware is actively attacking the system. Scaife et al. [18] built upon this mechanism as well, and produced an early-warning detection system. They evaluated their system against 14 ransomware families. By detecting file system changes characteristic of ransomware they were able to detect 100% of the ransomware families, however, detection only happened after a median loss of 10 files before detection. They additionally tested their system for false-positives against 30 benign applications, of which only 1 produced a false-positive. Using similar insights, Continella et al [25] developed ShieldFS, a driver that profiles the typical activity of a Windows system and can identify anomalous I/O activity generated by ransowmare, allowing the system to roll back the malicious changes. ShieldFS was developed concurrently and independently of PayBreak.

A framework to promote awareness and to educate individuals about the ransomware threat was proposed by Luo et al. as a proactive prevention solution [36]. Although teaching users how to safely and securely access the Internet can decrease the likelihood of a ransomware infection through spam and drive-by downloads, it does not eliminate the threat. As we have seen with the SamSam family (§5.2), ransomware now has worm capabilities to self propagate and infect vulnerable machines. In the case of zero-day vulnerabilities, no amount of education will protect a user.

Previous work has investigated methods to identify cryptographic functions within a binary. Calvet et al. developed a technique to achieve this within obfuscated programs and evaluated the system on known malware samples [24]. Although their technique could be a valuable asset for researchers to identify ransomware's encryption functions, the poor performance makes it impractical to include in an online defense. Furthermore, Caballero et al. [22] and Wang et al. [45] developed approaches to obtain plain text from encrypted communications.

Of the previously discussed work, PayBreak is the only solution providing an online protection scheme against ransomware, allowing a victim to fully recover their ransomed files. The negligible performance overhead proves our defense to be a practical solution.

8. CONCLUSION

PayBreak is a novel protection mechanism that defeats the threat of crypto-based ransomware. Early ransomware families failed because of their incorrect use of cryptographic functionality. Successful families switched to using the correct cryptographic approach – hybrid encryption. We identify that the symmetric session keys have to be used on the victim's host to perform file encryption. Thus, PayBreak implements a key escrow mechanism that stores session keys in a key vault. Keys in the vault are encrypted with the user's public key and thus, only the user's private key can unlock the vault. As opposed to government-mandated key escrow systems, PayBreak ensures that only the legitimate user has access to the keys held in escrow. We evaluated PayBreak on 107 ransomware samples and demonstrate that PayBreak can successfully recover from the damage caused by twelve different ransomware families. Furthermore, the runtime overhead of PayBreak is far below the human perception threshold and thus allows the use of PayBreak on production systems. Lastly, PayBreak will be released as a publicly available open source project.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments and our shepherd Guofei Gu for helping us improve the quality of this manuscript. This paper was supported by an EPRSC-funder Future Leaders in Engineering and Physical Sciences Award, by the EPSRC under grant EP/N008448/1, and by the Office of Naval Research under grant N00014-15-1-2948 Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

9. REFERENCES

[1] Alma ransomware: Analysis of a new ransomware threat (and a decrypter!). https://info.phishlabs.com/blog/alma-ransomware-analysis-of-a-new-ransomware-threat-and-a-decrypter.

[2] Cryptowall, teslacrypt and locky: A statistical perspective. http://blog.fortinet.com/post/cryptowall-teslacrypt-and-locky-a-statistical-perspective.

[3] Cybercriminals rake in $325m from cryptowall ransomware: report. http://www.washingtontimes.com/news/2015/nov/2/cybercriminals-rake-in-325m-cryptowall-ransomware/.

[4] Downloading and using the trend micro ransomware file decryptor. https://success.trendmicro.com/solution/1114221.

[5] Dxxd ransomware decrypter. https://github.com/eugenekolo/dxxd-decrypter.

[6] Fbi suggests ransomware victims to pay ransom money. http://thehackernews.com/2015/10/fbi-ransomware-malware.html.

[7] Ida f.l.i.r.t. https://hex-rays.com/products/ida/tech/flirt/.

[8] Kaspersky announces 'death' of coinvault, bitcryptor ransomware. http://www.theregister.co.uk/2015/11/02/kaspersky_announces_death_of_coinvault_bitcryptor_ransomware/.

[9] Pokemongo ransomware comes with some clever tricks. https://blog.malwarebytes.com/threat-analysis/2016/08/pokemongo-ransomware-comes-with-some-clever-tricks/.

[10] Ransomware. http://www.trendmicro.com/vinfo/us/security/definition/ransomware#List_of_Known_Ransomware_Families.

[11] Remove gerkaman@aol.com ransomware. http://www.virusresearch.org/remove-gerkamanaol-com-ransomware/.

[12] Researchers break encryption of marsjoke ransomware. http://www.securityweek.com/researchers-break-encryption-marsjoke-ransomware.

[13] The story of yet another ransom-fail-ware. http://esec-lab.sogeti.com/posts/2016/06/07/the-story-of-yet-another-ransomfailware.html.

[14] Themida. http://www.oreans.com/themida.php.

[15] This weird ransomware strain spreads like a virus in the cloud. https://blog.knowbe4.com/new-virlock-ransomware-strain-spreads-stealthily-via-cloud-storage.

[16] Threat spotlight: Teslacrypt – decrypt it yourself. http://blogs.cisco.com/security/talos/teslacrypt.

[17] Zynamics bindiff. https://www.zynamics.com/bindiff.html.

[18] Cryptolock (and drop it): Stopping ransomware attacks on user data. In *IEEE 36th International Conference on Distributed Computing Systems*, 2016.

[19] J. Berdajs and Z. Bosnić. Extending applications using an advanced approach to dll injection and api hooking. volume 40, pages 567–584. John Wiley & Sons, Ltd.

[20] M. Bourquin, A. King, and E. Robbins. Binslayer: Accurate comparison of binary executables. In *Proceedings of the 2Nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop*, PPREW '13.

[21] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *Usenix security symposium*, 2011.

[22] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *ACM conference on Computer and communications security (CCS)*, 2009.

[23] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. RFC2440: OpenPGP Message Format, 1998.

[24] J. Calvet, J. M. Fernandez, and J.-Y. Marion. Aligot: cryptographic function identification in obfuscated binary programs. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

[25] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi. Shieldfs: a self-healing, ransomware-aware filesystem. In *Annual Computer Security Applications Conference (ACSAC)*, 2016.

[26] Darren Pauli. Cryptowall 4.0: Update makes world's worst ransomware worse still - The Register. http://www.theregister.co.uk/2015/11/09/cryptowall_40/.

[27] M. Egele, M. Woo, P. Chapman, and D. Brumley. Blanket execution: Dynamic similarity testing for program binaries and components. In *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[28] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt. Bringing science to digital forensics with standardized forensic corpora, 2009.

[29] A. Gazet. Comparative analysis of various ransomware virii. *Computer virology*.

[30] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, et al. Manufacturing compromise: the emergence of exploit-as-a-service. In *ACM conference on Computer and Communications Security (CCS)*, 2012.

[31] F. Gröbert, C. Willems, and T. Holz. Automated identification of cryptographic primitives in binary programs. In *RAID*, volume 6961, pages 41–60, 2011.

[32] High-Tech Bridge Security Research. RansomWeb: emerging website threat that may outshine DDoS, data theft and defacements? https://www.htbridge.com/blog/ransomweb_emerging_website_threat.html.

[33] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda. Unveil: A large-scale, automated approach to detecting ransomware. In *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, Aug. 2016. USENIX Association.

[34] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda. Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, volume 9148 of *Lecture Notes in Computer Science*, Milan, Italy, July 2015. Springer International Publishing.

[35] P. Lestringant, F. Guihéry, and P.-A. Fouque. Automated identification of cryptographic primitives in binary code with data flow graph isomorphism. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, 2015.

[36] X. Luo and Q. Liao. Awareness education as the key to ransomware prevention. *Information Systems Security*, 2007.

[37] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277. ACM, 1968.

[38] D. Nazarov and O. Emelyanova. Blackmailer: The story of gpcode. https://securelist.com/analysis/publications/36089/blackmailer-the-story-of-gpcode/.

[39] B. Ramsdell. RFC3851: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification, 2004.

[40] G. Saito and G. Stringhini. Master of puppets: Analyzing and attacking a botnet for fun and profit. *arXiv preprint arXiv:1511.06090*, 2015.

[41] K. Savage, P. Coogan, and H. Lau. The evolution of ransomware. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf.

[42] B. Schneier. Memo to the Amateur Cipher Designer.

[43] M. I. Sharif, A. Lanzi, J. T. Giffin, and W. Lee. Impeding Malware Analysis Using Conditional Code Obfuscation. In *Proceedings of Network and Distributed Systems Security Symposium (NDSS)*, San Diego, CA, Feb. 2008. Internet Society. bibtex:sharif:conditional-code-obfuscation.

[44] L. Sun. Reform: A framework for malware packer analysis using information theory and statistical methods, 2010.

[45] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. Reformat: Automatic reverse engineering of encrypted messages. In *Computer Security–ESORICS 2009*, pages 200–215. Springer, 2009.

[46] A. Young and M. Yung. Cryptovirology: Extortion-based security threats and countermeasures. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 1996.

[47] A. Young and M. Yung. *Malicious cryptography: Exposing cryptovirology*. John Wiley & Sons, 2004.

[48] A. L. Young and M. M. Yung. An implementation of cryptoviral extortion using microsoft's crypto api. 2005.

APPENDIX

A. RANSOMWARE PSEUDOCODE

```
c2 = ConnectToCommandAndControl();
// Private key kept secret on C2
pubkey = c2.ReceivePubKey();
hPubkey = CryptImport(pubkey);
hCsp = CryptAcquireContext();
while (filename = FindNextFile()) {
    // Read
    ptFile = ReadFile(filename);
    // Generate random session key per file
    hSymkey = CryptGenKey(hCsp);
    // Then encrypt
    ctFile = CryptEncrypt(hSymkey, ptFile);
    keyblob = CryptExportKey(hPubkey, hSymkey);
    DeleteFile(filename);
    // Write encrypted session key
    WriteFile(filename, keyblob);
    // Append the encrypted file
    AppendFile(filename, ctFile);
}
```