Explorations of Ransomware

Mohannad Usama Abunassar
mabunass@mit.edu

Daniel Prado Sánchez
pradosan@mit.edu

Yousif Farid Dawood Rouben
yousif@mit.edu

María Ximena Rueda Guerrero
mxruedag@mit.edu

May 16, 2018

1 Introduction

Ransomware is a form of malicious software that prevents users from accessing their system until they pay a ransom. The outline for a typical attack is as follows: (1) Infection, (2) Encryption, (3) Extortion, (4) Decryption [1], where encryption is executed using a key that is inaccessible to the user and must be obtained from the attacker. Ransomware attacks have gained a lot of popularity over the last year, affecting systems that contain critical data such as health care providers, government entities and multinationals, and also bringing great profits for attackers [1]. For example, in 2018 the city of Atlanta was hit by a costly ransomware attack [2]. As ransomware victims invest large amounts of resources to prevent and mitigate these attacks and recover data, we believe it is important to explore variations of this malware to understand the possible risks that systems are exposed to.

This paper proposes two novel variants of ransomware attacks, where each provide an alternative to ransom payments in order to regain access to their system. The first, RansomShare, enables a user to decrypt their files once they have successfully infected a minimum number of other users. The second, Detention, enables a user to decrypt their files by computing a solution to a puzzle that is guaranteed to take a predetermined bounded amount of time, with the user having the option to pay chosen amounts to reduce this time proportionately.

The remainder of this paper is divided into four sections. Section 2 reviews the relevant schemes used to implement a proof of concept for the two proposed versions of ransomware. Section 3 describes the methodologies of construction for both variants of ransomware. Section 4 describes

the procedures used for testing the two implementations. Finally, Section 5 concludes with remarks on the potential uses, limitations and extensions for both Detention and RansomShare.

2 Background

In this section, we briefly review two key schemes, Shamir Secret Sharing and Time Lock Puzzles, that form the basis of obfuscation and recovery for RansomShare and Detention respectively.

2.1 Secret Sharing

Threshold schemes are useful for dividing up some secret S into n shares such that two properties hold:

1. It it possible to reconstruct the secret with k or more shares where $k \leq n$.
2. Any combination of less than k shares reveals nothing about S.

Shamir Secret Sharing [3] is a (k, n) threshold scheme. The scheme generates a random $k-1$ degree polynomial,

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_{k-1} x^{k-1}$$

and hides the secret S as the constant term in the polynomial, such that

$$f(x_0) = S.$$

The scheme then distributes n distinct points (x_i, y_i) along the polynomial $f(x)$, where $i \in [1, n]$. Since the secret as shown above is evaluated at $x = 0$, (secret = the constant coefficient of order zero = a_0), no shareholder should receive the point $(x = 0, a_0)$.

This scheme is used by RansomShare in order to allow an infected victim to solve for a secret which can be exchanged for their corresponding secret key. For each additional victim they infect, a victim acquires a unique share to their personalized secret. For a k solvable Shamir Secret Sharing scheme, only when $t \geq k$ shares have been acquired, can a victim correctly solve for their secret. For the purposes of this paper, we set $k = n$.

2.2 Time-Lock Puzzles

Time Lock Puzzles are designed using cryptographic properties with the goal of preventing a solver from using parallel or distributed computing to solve the problem faster. This is done by forcing

the solution of the problem to be intrinsically sequential. For this paper, we focus on using the general approach described by Rivest et al. [4] that utilizes the forcibly sequential nature of solving a large squaring problem.

A secret message M is first encrypted using key K, where K needs to be sufficiently long such that guessing is infeasible. For the purpose of this paper, the `AES` encryption standard is used, since the `RC5` scheme used in the original paper has been broken. Next, K is locked such that a squaring problem must be solved. The difficulty of such a problem is dictated through a solver's throughput in calculating squarings per second and finally, the number of seconds that the puzzle is intended to take to solve. In order to decrypt M, key K must first be unlocked. For the purposes of this paper, the secret message M will be the `AES` key used to encrypt a file system, while the remainder of the scheme remains unchanged.

Firstly, a composite modulus n is generated as a product of two large randomly generated primes p and q.

$$n = pq.$$

A sufficiently long `AES` key, K, is then used to encrypt the message M and produces a ciphertext C_M,

$$C_M = Enc_K(M).$$

Then, a random a modulo n where $1 \leq a \leq n$ is picked so that K is locked as follows,

$$C_K = K + a^{2^t} (mod\ n),$$

where

$$t = TS,$$

and T is the number of seconds required to solve the Time Lock Puzzle and S is the number of squarings modulo n that can be performed by the solver per second. Locking K more efficiently requires the following computations:

$$\phi(n) = (p-1)(q-1),$$

$$e = 2^t (mod\ \phi(n)),$$

$$b = a^e (mod\ n).$$

At this point, the Time Lock Puzzle is generated and represented as (n, a, t, C_K, C_M).

Since p and q are not given to the solver, solving the puzzle in the efficient method described above would require factoring n such that $\phi(n)$ can be computed. However, it is known that factoring n is a computationally hard problem. As a result, the fastest way to unlocking the puzzle reduces to a repeated squaring problem, which has been noted to be intrinsically sequential, thus enforcing some guarantees on the solve time bounds.

2.3 Secure Communication: Riposte

Although secure communication is outside of the scope of this project, this section briefly discusses one method by which this could be achieved. This is important, because the execution of our ransomware attacks requires communication over a network between the attacker and the victim's machine, while protecting the anonymity of the attacker. In order to ensure this and provide traffic-analysis-resistance while scaling widely on the network, we propose using Riposte. Riposte is a messaging system developed at Stanford University that offers a number of security and scalability properties that are essential for the success of our ransomware attacks. Firstly, Riposte protects against traffic analysis attacks which is vital for maintaining the anonymity of the network which the attacker communicates over with the victims' machines. To ensure staying anonymous on a larger scale, deploying three different servers (three non-colluding attacker machines) for the attacker can guarantee building an anonymity set of about 3 million machines in 32 hours. It is important to note that the anonymity of users over anonymous networks like Tor can be broken using certain traffic analysis attacks as several research papers have shown which means that Riposte offers more resilience to these types of attacks than other existing anonymous networks. Secondly, Riposte prevents malicious clients from anonymously executing denial-of-service attacks which allows the attacker to easily detect and exclude unwanted requests from any machine. Finally, Riposte allows scaling up to anonymity set sizes of millions for certain latency-tolerant applications. This will allow ransomware attacks to spread out to large number of machines while still maintaining the security features offered by Riposte. Since ransomware already uses standard public-key encryption to send a message from attacker to victim, point-to-point private messaging channels can be easily established while using Riposte [5].

3 Methodology

This section provides a breakdown of the different components constituting the RansomShare and Detention schemes. First, the mechanism by which the file system is encrypted and decrypted is outlined. Next, the key components of RansomShare are discussed starting from victim infection

and identification to malware redistribution and secret exchange which operate under the guise of Shamir Secret Sharing. Finally, the procedure by which Detention is mounted on a victim's machine is described along with a brief summary of the communication and key recovery protocols.

For the remainder of this section, the master (original attacker's) machine is referred to as `MASTER`.

3.1 Skeleton File-System Encryption Scheme

The encryption scheme developed uses Advanced Encryption Standard `(AES)` to encrypt the files on the victim's machine. The `AES` session cipher that will be used to encrypt the files gets generated locally on the victim's machine. In order to protect this key from being exposed, it gets encrypted on the victim's machine. In RansomShare, `MASTER` generates `RSA` key pairs associated with each victim where the `RSA` public key is used to encrypt the `AES` session cipher once all target files are encrypted. As for Detention, the `AES` cipher is encrypted using another `AES` key which is locked with a time lock puzzle. The `AES` encryption is done in Cipher Feedback Mode `(CFB)` which makes a block cipher into a self-synchronizing stream cipher. `CFB` allows us to encrypt any file regardless of its size as the block size is dynamically chosen to fit the size of the data to be encrypted. Since the attacker uses the same `AES` key for encrypting more than one file, the attacker needs to generate different Initialization Vector values `(IV)` for each file it encrypts. The `IV` is a 16 bytes long value that is randomly generated and stored at the end of each encrypted file to be used later on for decryption. The way the `IV` is generated and stored in each file could be altered to make it hard for an adversary to know the values used. Note that a ".enc" extension gets added to the end of every file once it gets fully encrypted, as a means of indicating which files have been encrypted.

3.2 Skeleton File-System Decryption Scheme

The scheme used to decrypt the encrypted files on a victim's machine is, as expected, analogous to the encryption scheme developed. Decryption is performed on files ending with ".enc" and that extension gets removed when decryption is completed. The first step for decrypting the file system requires decrypting the `AES` key that was used for encrypting the file system on the victim's machine. In RansomShare, this is done using the `RSA` secret key `SK` that belongs to that victim. In Detention, this requires solving the Time Lock Puzzle in order to get the `AES` key K_2 that was used to encrypt the encryption `AES` key K_1. K_2 will then be used to decrypt K_1 which in turn will be used to decrypt the file system. Once K_1 is decrypted, it is used along with the `IV` value that was used for encrypting a file, which gets retrieved by reading the last 16 bytes of every file.

From this, an AES decryption cipher is generated which is then used in CFB mode to decrypt the file. This procedure is repeated for all encrypted files.

3.3 RansomShare: Shamir Secret Sharing Approach to Key Security

RansomShare follows the regular file encryption scheme outlined in section 3.1. Once a victim has been infected, they then have the option of either acquiring the key from MASTER using some third party payment system or infecting n other machines. RansomShare's steps can broken down as follows: (1) Initial payload communication (Figure 1a), (2) Victim identification (Figure 1b), (3) Spread (Figures 1c, 1d), (4) Secret Exchange (Figure 1e, 1f). For the remainder of this section, the victim machines' are denoted by V_i for $i \in Z^+$.

Whenever one victim V_1 infects another victim V_2, V_2 will begin their own process of following steps 2-4.

3.3.1 Initial Payload Communication

This step consists of MASTER communicating the payload to a victim V_1. If V_1 is convinced to open the payload, then RansomShare will begin its process of infection.

3.3.2 Victim Identification

After V_1 executes the payload, the payload will communicated with MASTER and report that a new infection has occurred. Since V_1 was infected by MASTER, there is no previous unique identifier to send to MASTER. MASTER then generates a new unique identifier (UID_1) and RSA public, private key pair (PK_1, SK_1), but only responds to the initial infection with UID_1 and $PK1_1$. The payload then encrypts V_1's file system as described in section 3.1.

3.3.3 Spread

Once V_1 has been infected, they are presented with two recovery options. One is to use third party payments to retrieve SK_1 from MASTER. The other is to infect n other machines. In order to do this, V_1 needs to send the payload along with UID_1 to another victim V_2.

If V_2 were to run the payload sent to it by V_1, the victim identification step begins but V_2 sends UID_1 as the previous unique identifier (this informs MASTER that V_1 is the direct cause of this infection).

When a successful infection occurs, the next time V_1 requests its shares, MASTER will respond with one more share than before.

V_1 will repeat this process until there are n successful infections as it will receive n total shares.

3.3.4 Secret Exchange

Once V_1 has n shares it will solve for the Shamir secret that corresponds to the given shares. V_1 can then exchange this secret for SK_1 which it can then use to decrypt its file system as outlined in section 3.2.

3.4 Detention: Time Lock Puzzle approach to Key Security

The Detention payload consists of a file system encrytor/decryptor, puzzle generator, communicator and a universal `RSA` public key `UPK` that is generated by `MASTER` and is static across all payloads. This public key, is used to secure all sensitive communications from victim to `MASTER`.
Detention's steps can broken down as follows: (1) Mounting the Payload (Figures 2a, 2b), (2)Communication with `MASTER` (Figures 2c, 2d), (3) Solving the Time Lock Puzzle (Figure 2e).

3.4.1 Mounting the Payload

Detention, once executed, attempts to load a valid Time Lock Puzzle checkpoint file from the file system. If none exists, this implies that either the ransomware is being executed for the first time, or it crashed before a puzzle could be saved. In this scenario, a new 32 bit random unique identifier `(UID)` is generated to identify the victim to `MASTER`, in addition to a new 256 bit random `AES` key K_1 used for encrypting and eventually decrypting the file system as described in sections 3.1 and 3.2. Next, the file system structure tree with absolute paths pointing to all files is extracted and a new 256 bit random `AES` key K_2 is generated, which is used to encrypt K_1. Then, K_2 is Time Locked and the resulting Time Lock Puzzle is saved as a checkpoint file. Subsequently, during puzzle generation, a secret triplet file `ST` consisting of $(Enc_{UPK}(K_1), Enc_{UPK}(K_2), Enc_{UPK}(UID))$ is also stored as a secret communications checkpoint file, while all unencrypted sensitive information (i.e. K_1, K_2 and Time Lock variables) that would enable a victim to directly decrypt their file system is deleted from memory. Next, two processes are started; one handling communication with `MASTER` and the other for solving the Time Lock Puzzle.

3.4.2 Communication with `MASTER`

The victim's machine, once infected, will attempt to communicate with `MASTER` for two purposes: transmitting encrypted secret infection data represented by `ST` and requesting easier puzzles that master will generate based off of the parameters in `ST` and a payment made by the victim.

The communication process begins by trying to load the secret triplet checkpoint file `ST`. If no such file exits, then communication of `ST` to `MASTER` is assumed to have been successfully completed in the past. If, however, said file exists, then it is loaded and post requests are sent to `MASTER` until confirmation of receipt is received, at which point the `ST` checkpoint file is cleared from the file system.

Once communication of `ST` has been verified, the victim periodically requests easier puzzles from `MASTER`. A request is successful if and only if `MASTER` receives a payment with a valid `UID`, at which point a new puzzle with difficulty proportionate to

```
(TOTAL HISTORICAL PAYMENTS + CURRENT PAYMENT)/MAX PAYMENT AMOUNT
```

is generated. This is only possible, however, due to the information necessary for generating new puzzles being communicated to master through the transmission of `ST`. Once an easier puzzle is transmitted in response to a corresponding payment, `MASTER` updates its internal state and the victim continues to periodically request easier puzzles, receiving one only when another payment is made.

3.4.3 Solving the Time Lock Puzzle

The solver process uses a custom-built repeated squaring mechanism that enables for checkpoints to be saved and estimated time to completion to be calculated and displayed at regular intervals. This ensures that if Detention crashes the victim can continue solving the puzzle without having to restart the computation at the original puzzle difficulty.

Once the puzzle is solved, Detention unlocks K_2 and then decrypts K_1 and proceeds to decrypt the file system as described in Section 3.2.

4 Experimentation and Results

RansomShare and Detention were tested by having them only encrypt a subset of files of varying sizes on the infected machines. Although the functionality allowed for detection and encryption of the entire file system tree, for the purposes of testing, controlled directories were used. The remainder of this section outlines the testing methodology for both ransomware schemes and the observed performance.

It should be noted that all of the experiments detailed in this section were tested both directly with the Python scripts and also with operating system specific one-file self-contained executa-

bles that were generated using the PyInstaller package. Generating these executables allowed for mounting attacks on systems that did not necessarily have all the required system dependencies.

4.1 RansomShare

RansomShare was successfully tested with varying number n of secret shares required to construct the Shamir secret. For the purposes of this paper, experiments were conducted on $n \in [1, 6]$, however the implementation is capable of handling arbitrarily large values of n. The initial payload was generated on `MASTER` and distributed to the first victim V_1. In these experiments, V_1 distributes the payload along with the its UID_1 to the n other machines set up for the experiment.

The payload was then executed on the newly targeted victim machines resulting in infection. V_1 was then able to decrypt their files only when n new victims were infected and successful commincation was established between all parties involved and `MASTER`.

4.2 Detention

The two key components of Detention are the time puzzle difficulty tuning and the recovery of incomplete puzzles.

The system was tested to completion after making zero, one, and several payments. The implementation included an extra feature that displayed a progress update to see the change in the expected time remaining to encryption based on the expected number of squarings the machine could compute. For times longer than ~ 15 minutes, this feature was used to verify that the expected time until completion was reduced proportionally to the aggregate payment made to `MASTER`.

In all cases, the execution was run both with and without interruptions to test the recovery capability that saves a partial solution to the puzzle. Simulations of communication failures between `MASTER` and the victims were also performed and this did not alter the behavior of the process running the time lock puzzle.

5 Conclusion

RansomShare is a powerful novel ransomware as it coerces trusted users to become part of the scheme. This can be a tremendous benefit to distributors of ransomware as potential victims may be more susceptible to opening a file if it comes from a user they believe to be trustworthy. However, RansomShare does have its limitations. In its current state, it relies on trusting the

fact that new connections are new infections. A fairly simple way to exploit this is to simply spin up multiple empty instances to be infected. This can then lead to a real victim thwarting RansomShare and retrieving their files without ever infecting a machine that may pay the ransom or has the potential of spreading RansomShare to even more machines. One way this can be mitigated is by limiting the original victim to only receive an additional share from an infection if that infection went ahead and paid the ransom. Such a modification would ensure that payments must eventually be made by direct neighbours and as such is a sufficient verification method.

Unlike RansomShare, Detention provides a more standalone approach to ransomware. Even if communications are cut off between the victim and `MASTER`, the ransomware can still be thwarted by solving a Time Lock Puzzle. Additionally, providing victims with the ability to make incremental payments to reduce unlock times proportionally leads to a probability distribution over payment values versus a typical nothing or all scheme, thus having potential impacts on the expected value of return for the attacker. One powerful extension to Detention would be to capitalize on the time factor of the Time Lock Puzzle. Since Detention is expected to be executed on a victim machine such that the solver works away at solving the puzzle, this means that an attacker could incorporate additional processes into the payload that would execute wanted code and thus leach off of a victims compute power. This could be used for varying purposes such as Bitcoin mining, anonymous routing schemes and spam generation.

6 Acknowledgements

We would like to thank the 6.857 teaching staff for providing us with the necessary support and guidance throughout the semester, which provided us with the foundations for developing the schemes described in this paper. In particular, we would like to thank Professor Ronald Rivest for his mentorship and feedback related directly to this project.

References

[1] M. Conti, A. Gangwal, and S. Ruj, "On the economic significance of ransomware campaigns: A bitcoin transactions perspective," *arXiv preprint arXiv:1804.01341*, 2018.

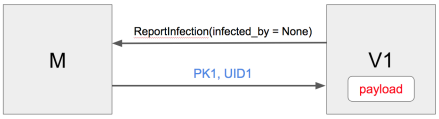
[2] L. H. Newman, "Atlanta spent $2.6M to recover from a $52,000 ransomware scare," *Wired*, Apr 2018.

[3] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

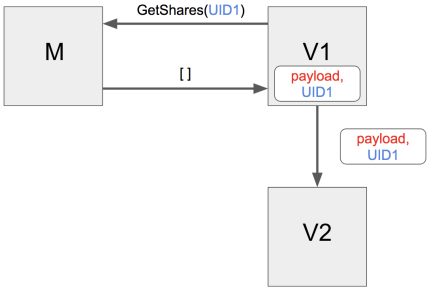
[4] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.

[5] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *Security and Privacy (SP), 2015 IEEE Symposium on*, pp. 321–338, IEEE, 2015.

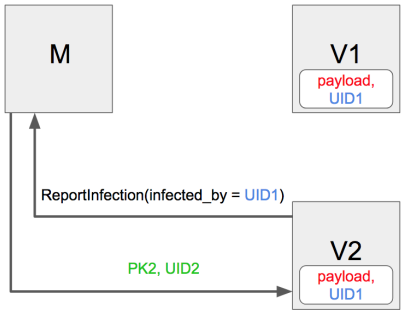
(a) `MASTER` sends the *payload* to a victim `V1`.



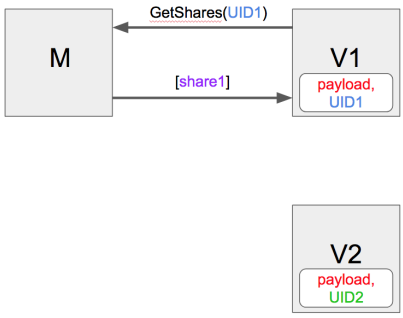
(b) `V1` opens a communication channel with `MASTER` to report that it has been infected. Since `V1` received the *payload* directly from `MASTER`, it does not report an infection source. `MASTER` sends `V1` the public key `PK1` that it shall use and its identifier.



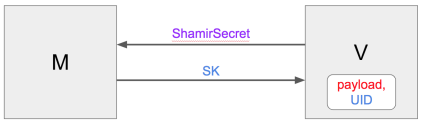
(c) `V1` periodically requests the shares to complete the secret to `MASTER`, who temporarily sends back no shares since it has received no reports of victims infected by `V1`. Simultaneously, `V1` sends a new victim, `V2`, the payload along with `UID1`.



(d) `V2` executes the payload received from `V1` and opens a communication channel with `MASTER` to notify that it has been infected by `V1`. `MASTER` sends `V2` its corresponding public key and identifier.



(e) `V1` requests shares from `MASTER` again, and obtains one share, since `MASTER` has received exactly a notification of exactly one victim infected by `V1`.

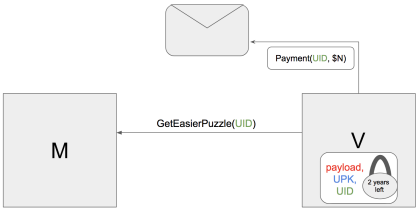


(f) After `V1` has successfully infected n victims, it solves for its Shamir secret and sends it to `MASTER`. `MASTER` verifies the correctness of the answer, and then sends `V1` the secret key to decrypt their files.

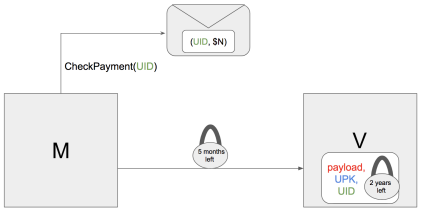
Figure 1: RansomShare Flow Chart

(a) `MASTER` sends the *payload* to a victim `V`.

(b) `V` executes the payload which generates a unique identifier `UID`, a key K_1 to encrypt their files, a key K_2 to encrypt K_1, and a time lock puzzle that solves to K_2. The victim reports to `MASTER` that it has been infected and sends over K_1, K_2 and its identifier. After successfully sending this to master, `V` deletes both keys.



(c) `V` makes a payment to a third party payment manager that includes its unique identifier, and requests an easier puzzle to `MASTER`. It then makes a request for an easier puzzle to `MASTER`.



(d) `MASTER` confirms that a payment has been made on behalf of `V`. `MASTER` proceeds to send `V` a new puzzle that is proportionally easier to the total payments made so far by `V` that also solves to K_2.

(e) Upon completion of (approximately) the time lock puzzle due date, the puzzle reveals K_2, which is used to decrypt K_1, and consequently the entire file system.

Figure 2: Detention Flow Chart