RDS3: Ransomware Defense Strategy by Using Stealthily Spare Space

Kul Prasad Subedi[‡], Daya Ram Budhathoki[‡], Bo Chen[§], Dipankar Dasgupta[‡]
[‡]Department of Computer Science, University of Memphis, Memphis, TN, USA
[§]Department of Computer Science, Michigan Technological University, Houghton, Michigan, USA
Email: {kpsubedi,dbdhthki,ddasgupt}@memphis.edu, bchen@mtu.edu

***Abstract*—Ransomware attacks are becoming prevalent nowadays with the increased use of crypto-currencies. As the most harmful variant of ransomware, crypto-ransomware encrypts the victim's valuable data, and asks for ransom money. Paying the ransom money, however, may not guarantee recovery of the data being encrypted. Most of the existing work for ransomware defense focuses on ransomware detection. A few of them consider data recovery from such attacks, but they are not able to defend against ransomware which can obtain a high system privilege.**

In this work, we design RDS3, a novel Ransomware Defense Strategy, in which we Stealthily back up data in the Spare space of a computing device, such that the data encrypted by ransomware can be restored. Our key concept is that the unused space can backup critical data are fully isolated from the system (which can be affected by Ransomware). In this way, the ransomware is not able to "touch" the backup data regardless of what privilege it can obtain. Security analysis and experimental evaluation show that RDS3 can mitigate ransomware attacks with an acceptable overhead.

I. Introduction

Ransomware attacks suddenly became very prevalent in recent years with the flourishing of crypto-currencies like bitcoin. Due to their anonymity, crypto-currencies offer ransomware makers a great mean of receiving ransom money without being identified. A recent security incident caused by a ransomware attack infected 900 systems used by the San Francisco Municipal Transportation Agency [6]. WannaCry ransomware attacked more than 300,000 computers from 99 countries [13]. According to FBI [8], ransomware has become a billion dollar a year crime and is still under continuous growth.

Similar to malware, ransomware utilizes all types of means (e.g., spam emails, mal-advertisements, social engineering) to propagate to a victim computing system. Then, it will either lock the victim's system (i.e., **locker ransomware**) or encrypt the data (i.e., **crypto-ransomware**) in the victim's system. Finally, it will require the victim to pay the ransom money in order to unlock the system or obtain the key for decrypting the data. According to the recent Internet Security Threat Report [4], crypto-ransomware has now dominated the ransomware family. Therefore in this paper, we mainly focus on defending against crypto-ransomware.

Paying the ransom money [7, 11] may not provide the victim a guarantee to obtain the actual key for decrypting the data attacked by ransomware. Even worse, this provides incentive to the ransomware makers to improve their ransomware and launch more advanced ransomware attacks. The existing research of ransomware [16, 21, 25, 32] mainly focused on designing effective ransomware detectors. Their ultimate goal is to detect ransomware in a timely manner such that the system can block ransomware before it causes more damage to the victim's data. Most ransomware detectors [21, 25, 32] rely on dynamic analysis [33], which usually allows the ransomware to run in order to observe the abnormal behavior. This unfortunately implies that regardless of the effectiveness of the detectors, a few victim's data are always encrypted by the ransomware before it is detected and blocked. In other words, a recovery component seems indispensable in ransomware defense.

The recovery component requires creating backup data, such that the data encrypted by ransomware are always recoverable. In general, if the victims periodically back up their data using external storage media or public cloud services, the ransomware attack would not even become an issue [5]. However, most people today are reluctant to back up their data for potential ransomware attacks due to three concerns: First, people usually do not even think they will become ransomware victims until they are really attacked; second, periodically backing up data will create additional work burden; third, backing up data in external storage media or public cloud services requires the users to plan additional budget in purchasing hardware equipment or cloud services. If a cloud storage service is used, an additional expense will be also required for Internet access.

The first concern can be mitigated by raising the awareness of ransomware attacks through education or media campaigns. The second concern can be resolved by automating the backup process such that the users can be liberated from such a burden. This work, however, aims to address the third concern, which is the most challenging one. Having observed that most computing devices usually possess a certain amount of spare space, we designed RDS3, a Ransomware Defense Strategy, by taking advantage of Spare space to Stealthily store a certain amount of redundant data. This is advantageous, since it fully utilizes the existing computing resources to defend against ransomware, eliminating the need of purchasing additional unnecessary computing resources. RDS3 consists of two main components, *a detector component* and *a recovery component*. The detector component will monitor the computing device. Once the ransomware is detected, it will take action to block the ransomware and inform the recovery

component. The recovery component will periodically back up the data to the spare space. Once the ransomware has been detected by the detector, it will take action to restore the data corrupted by the ransomware.

To design RDS3, we faced two main issues: 1) How can we prevent ransomware from having access to the backup data even though it is able to obtain a high privilege? By escalating to a high privilege, the ransomware probably can have access to the backup data, and simply encrypts them to attack our design. Previous work [21] unfortunately cannot address this issue; and 2) How can we effectively utilize the spare space? This is a practical issue, since a computing device usually possesses a very limited amount of spare space, which is usually not able to hold a mirror copy of the entire data. To address the first issue, we separate the entire storage medium into a regular volume and a backup volume. The regular volume is managed by a regular user OS which is for daily use; the backup volume is managed by a light-weight OS, which only runs the small backup/recovery applications. Both OSes are further isolated in such a way that regardless of what privilege the ransomware can obtain in the user OS, it will not be able to have access to the backup volume. To address the second issue, our ideas are: first, we back up the data in an incremental manner by utilizing delta encoding; second, we offer flexibility to the user on determining what data will be backed up, by allowing the user to stealthily mark his/her important files. Here "stealthily" means that these "marks" are only recognizable by the recovery component rather than the ransomware. This is necessary as it can prevent ransomware from learning what data may be critical to the user.

Contributions. Our contributions can be summarized as follows:

- To the best of our knowledge, RDS3 is the first design for ransomware defense which contains both a detector component and a recovery component and, meanwhile, is able to defend against the ransomware that can obtain a high privilege (e.g., root privilege).
- RDS3 fully utilizes the existing resources in a computing device to achieve a certain level of security. This idea itself may possess independent interest.
- We analyze the security of RDS3. In addition, we implement RDS3 in a real-world system, and experimentally evaluate its performance.

II. BACKGROUND

A. Ransomware

Ransomware is a special type of malware whose sole purpose is to prevent victims from accessing their valuable data either by encrypting the data or locking the systems. By leveraging various attack means, e.g., spam emails, malicious advertisements, social engineering, SMS messages, Traffic Distribution System (TDS), etc., the ransomware gets itself installed in the victims computing system, unbeknownst to the victim. Later, the ransomware either locks the system or encrypts the data stored in the device, and extorts the victim for money in exchange for the key to unlock the system or decrypt the data. In this sense, the existing ransomware can be categorized into crypto-ransomware and locker ransomware.

The crypto-ransomware asks for ransom money by encrypting the victim's valuable data, while the locker ransomware locks the victim's device and demands for money in order to unlock it. Typical crypto-ransomware includes CryptoWall [20], CryptoLocker [24], and Locky [19], and typical locker ransomware includes Winlocker [31]. In comparison, the crypto-ransomware is much more harmful than the locker ransomware, since locker ransomware only locks the victim out of the system, and the victim can still have access to his/her data, e.g., by removing the storage medium from the infected computing system, and using it in another non-infected computing system to copy out the data. Crypto-ransomware uses cryptographic encryption algorithms to encrypt the victim's data and the key may be stored in a remote command-and-control (i.e., CC) server, rendering it difficult to recover the data being encrypted without paying the ransom money.

B. Isolation Techniques

Secure computation requires a secure processing environment which can ensure the protected resources (e.g., memory and peripherals) will not be tampered/eavesdropped by the adversary. This usually relies on isolation techniques, by which the protected resources will be completely isolated from the unsecure environment. In general, isolation can be achieved using either hardware or software. The hardware-based isolation includes ARM TrustZone, Intel Software Guard Extensions ,(SGX) etc.

The software-based isolation completely depends on the confinement provided by the layers of software or kernel. It can be achieved by various tools and techniques. One example is chroot, which provides some form of file system isolation for non-root processes. However, users having root privileges can easily escape from the chroot isolation. Another popular tool for isolation is software container, e.g., Docker [28] and Linux container [22]. The most popular way of achieving software-based isolation is via virtual machine monitors. A virtual machine (VM) is a virtual computer system, which can be viewed as an isolated independent software container with an operating system and applications inside. Multiple VMs can be put on a single physical computer, enabling multiple operating systems and applications to run on one single physical host computer. As a thin software layer stays between host operating system and the guest operation system, the virtual machine monitor can decouple the virtual machines from the host and dynamically allocate computing resources to each virtual machine as needed. Popular virtual machine monitors include VMware vSphere [12], Citrix XenServer [14], Microsoft Hyper-V [2], Oracle VirtualBox [30], etc.

III. SYSTEM AND ATTACK MODEL

System model. We mainly consider computing devices which are equipped with mass storage. These include servers, desktops, laptops, etc. We do not consider those computing devices which are equipped with limited resources, e.g., smart phones. This is because, we rely on the assumption that the computing

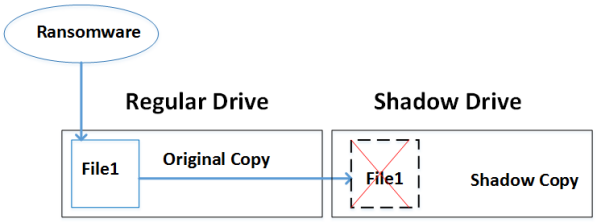

Fig. 1: An attack on ShieldFS

devices being protected should have enough empty space, which is not necessarily true for those devices (smart phones, smart watches, etc.) equipped with limited resources.

Attack model. We only consider crypto-ransomware which tries to encrypt the data in the victim device. After being able to get itself installed in a running environment, the ransomware is able to escalate its privilege to a level equal to the operating system which controls this environment (e.g., root privilege). This may be achieved by exploiting various system vulnerabilities [10, 23]. *We do not consider other types of malware which exhibit different attack behavior*.

IV. Attack Scenarios

The sole existing work for ransomware defense which includes both the detector component and the recovery component is ShieldFS [21]. ShieldFS automatically shadows a file to a shadow drive whenever it is modified, and such a shadow copy can be used to recover the file encrypted by the ransomware. However, if the ransomware can obtain high system privilege and can have access to the shadow copy, it can simply encrypt both the original copy and the shadow copy, rendering the file unrecoverable (see Figure 1). Although ShieldFS claims to make the shadow drive read-only to deny any modification request, this cannot work if the ransomware obtains high privilege (e.g., root privilege).

V. Our Design: RDS3

This section presents our main design, RDS3, a Ransomware Defense Strategy utilizing Spare space to Stealthily store a certain amount of backup data. Our design aims to defend against ransomware which can obtain high system privilege.

A. Key Insights

Before presenting our main design, we first describe our key insights by discussing a few questions below.

Question 1: why are backup data necessary for ransomware defense?

A main objective of ransomware defense is to restore the data encrypted by ransomware. To achieve such a goal, a straightforward solution could be to obtain the decryption key. This may be possible if the ransomware utilizes symmetric encryption, since the key was present in the victim computing device in the past which may be extracted [27]. However, such a solution cannot work when the ransomware directly or indirectly utilizes asymmetric encryption, which is unfortunately the case dominating the existing ransomware. For asymmetric encryption, computing the private key by brute force is usually not possible considering the large key space used by the ransomware (e.g., an RSA key is at least 1024-bit). Therefore, obtaining the decryption key without paying the ransom money seems infeasible. Without obtaining the decryption key, the only option for recovering data being corrupted by ransomware is by creating backup data.

Question 2: where to store the backup data?

Simply relying on external storage (e.g., a mobile disk or a cloud storage provider like Amazon S3) is not necessarily good, since it requires the user to pay for additional hardware or services, as well as bring additional burden to the user on maintaining such an additional medium. By observing that most of the computing devices usually possess a certain amount of spare space (see Section VI-B), we propose to utilize a portion of the spare space to store the backup data. This is advantageous, as it eliminates the need of purchasing additional storage media/services. Most importantly, for the first time, we design a secure system which can mitigate ransomware attacks by fully utilizing the remaining resources in a computing device.

Question 3: how to make the space which stores the backup data inaccessible by ransomware?

A question that remains unanswered is how to ensure that the ransomware is not able to encrypt the backup data stored in the spare space, regardless of its privilege. Intuitively, this can be achieved by restricting the privilege the ransomware can obtain, and meanwhile storing the backup data stealthily in the spare space. We introduce a regular volume and a backup volume. The regular volume is used for storage of regular data, while the backup volume is used for storage of backup data. Note that the backup volume is built using the spare space. To prevent the ransomware from having access to the backup volume by escalating its privilege, we leverage isolation techniques. Specifically, a user operating system is introduced to manage the regular volume, with all the applications for daily use. A light-weight operating system is introduced to manage the backup volume, with a few small applications simply supporting backup and recovery functionality. Then, an isolation technique is introduced to prevent the user OS from accessing the backup volume. This ensures that even though ransomware can compromise the entire user OS and obtain the root privilege, it is still not able to have access to the backup volume, as the backup volume is transparent to the user OS.

Question 4: when to back up the data?

By observing that ransomware always needs to modify the victim data (e.g., over-write the data with their encrypted version, or simply delete them which is also an over-write operation), ShieldFS [21] creates a shadow file copy each time when a modify operation is performed on a file. This may be problematic, as a piece of ransomware which is able to obtain root privilege can easily observe this special system behavior and intercept into the back up process to disturb the defense (e.g., corrupt the shadow copy). The aforementioned

issue is due to the fact that the back up process happens when ranswomare is already present in the system. To address this issue, we move the back up process backward, such that data will be backed up before the ransomware is present. This however, makes it challenging to determine what data need to be backed up, as the system has no knowledge about what data the ransomware will corrupt.

Question 5: what data will be backed up?

To defend against ransomware which can obtain root privilege, it is necessary to back up data before the ransomware is present in the victim system. However, this sacrifices the advantage of knowing what files the ransomware is "touching." Without such knowledge, the system is not able to identify those files which will be most likely corrupted by ransomware. One remediation could be to back up all the files in the regular volume, which is unfortunately infeasible considering that the spare space in a computing device is usually small. We address this issue from another angle. We select the files to be backed up based on their "importance" to the device owner. However, to identify whether a file is important or not is not straightforward. We believe a reliable way is to allow the device owner to make this decision, as he/she is the right person who clearly knows whether a file is important to him/her. Specifically, the system marks each newly created file as "unimportant" by default, and a system tool is provided to allow the device owner to re-mark any file as "important."

Note that we should prevent ransomware from learning the "importance" information. Otherwise, it may take advantage of this information to launch more advanced attacks, e.g., steal important files (which only incurs read operations and is difficult to be detected) and ask for ransom money by threatening to release them to public. To protect the "importance" information, we encrypt it via asymmetric encryption. Specifically, the "importance" information is encrypted using a public key. The ransomware is not able to obtain the corresponding private key, and is thus not able to decrypt this information. For different files, the same "importance" information will be encrypted into different cipher-texts, such that the ransomware is not able to learn whether any given files are in the same "importance" category. To achieve this, we embed the file ID into the encryption. In addition, we embed a large nonce when performing encryption, such that with negligible probability, the same cipher-text can be re-generated for the same file by knowing the file ID.

B. Design Details

We are now ready to present our main design, RDS3. Let $(\mathsf{KGen}, \mathsf{AEnc}, \mathsf{ADec})$ be a secure asymmetric encryption scheme.

The overall architecture of RDS3 is shown in Figure 2. We create two volumes, a *regular volume* and a *backup volume* over the underlying storage medium (e.g., *disk*). A *user operating system* is used to manage the regular volume for daily use, e.g., all the daily *user applications* are run within this OS, and all the user data are stored in the regular volume. A light-weight *tiny OS* is used to manage the backup volume for backup purposes. Only a few applications (e.g., *backup app* and *recovery app*) are run in this tiny OS, and only the backup data are stored in the backup volume. All the computing resources (e.g., storage space, memory) and components (e.g., user OS, backup OS) are managed by a virtual machine monitor. The isolation provided by the virtual machine monitor ensures that the user OS is not able to access the backup volume[1]. To simplify the notations, we use **VM1** for the user operating system, and **VM2** for the tiny OS for backup volume.

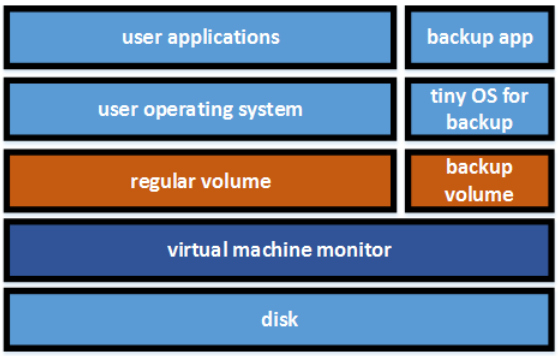

Fig. 2: The architecture of RDS3

RDS3 consists of two components: a *detector component* and a *recovery component*. The detector component runs in VM1. It constantly monitors VM1 and will notify the recovery component when ransomware is detected. The recovery component mainly runs in VM2. It periodically pulls data from the regular volume for back up purposes. Once having received the notification from the detector, it will work with VM1 to restore the data being protected. There are a large number of detectors available in the literature [16, 21, 25, 32] that monitor either system behavior or file access patterns, which can be simply adapted here for our detector component. To prevent the ransomware from disturbing the detector component, we should run this component in a privilege higher than the root privilege of VM1 (e.g., a privilege comparable to the that of the virtual machine monitor). In the following, we will only elaborate the design of the recovery component, which is the main focus of this paper.

1) Recovery Component: The recovery component consists of a backup application (i.e., backup app) and a recovery application (i.e., recovery app). The backup app periodically communicates with VM1 to pull data from the regular volume, and stores them to the backup volume. The recovery app is activated when ransomware is found and blocked. It will then work with VM1 to read data from the backup volume, to recover the data being corrupted by the ransomware in the regular volume. Note that VM1 is not able to have direct access to the backup volume due to isolation enforced by the virtual machine monitor.

Since the backup volume is created using the spare space in a computing device, it will be limited in capacity (e.g., 10%-20% of the entire storage capacity). To address this

[1]Here we assume the virtual machine monitor can provide a good isolation. Penetrating the barrier created by virtualization is out of our research scope.

concern, we do the following: first, we allow the device owner to mark his/her "important" files; second, when backing up the data to the backup volume, we utilize delta encoding [18] such that only the difference between two subsequent version is transmitted and stored. We will elaborate the key functionality of the recovery component, namely, *initialization*, *marker generation*, *back up data*, and *data recovery*.

Initialization. A key pair (pk, sk) is generated for asymmetric encryption by running $(pk, sk) \leftarrow \mathsf{KGen}(\lambda)$, where λ is a secure parameter. The public key pk is distributed to VM1, and the private key sk is kept secret in VM2. The device owner picks a credential and keeps it in a secure location.

Marker generation. We allow the device owner to mark the files as "important." These files will be periodically pulled by the backup app to store in the backup volume. To generate the marker, we face two challenges: 1) where to store the marker; 2) how to keep the marker secret from the ransomware. By observing that most file systems maintain a set of attributes for each file, we address the first challenge by storing the marker information as one additional file attribute. If the file is an "important" file, we simply set the marker as "1". Otherwise, the marker is "0". To address the second challenge, we encrypt the marker using asymmetric encryption. This is necessary, as the marker is generated in VM1. Using symmetric encryption in VM1 may create a risk of key leakage, since we cannot predict when the ransomware is present. Using asymmetric encryption only requires the public key to be present during encryption, which will mitigate the aforementioned risk. In addition, to prevent the ransomware from learning that two different files have the same mark, we embed the file ID into the encryption. A marker attribute for file f_1 will be computed as: $\mathsf{AEnc}_{pk}(marker\ for\ f_1||file\ ID\ for\ f_1||nonce)$. To ensure that the *file ID* is unique for each file, it can be generated by concatenating the ID of the computing device and the file name.The *nonce* is a large enough random number generated each time when computing a marker attribute.

The detailed process for marker generation is shown in Algorithm 1.

Algorithm 1 Marker generation

```
1: procedure GenMarker(pk, file_id, marker, λ)
2:     nonce ←R {0, 1}^λ
3:     ciphertext ← AEnc_pk(marker||file_id||nonce)
4:     attribute ← Base64Encode(ciphertext)
5:     ret ← SetFileAttribute(attribute)
6:     return ret
```

When a file is created, the system will mark it as "0" by default. This can be achieved by running $\mathsf{GenMarker}(pk, file\_id, 0, \lambda)$. In a Unix based system, inotify API provides a mechanism to monitor filesystem events [3]. When a new file is created, inotify triggers GenMarker to set the default marker.

Back up data. Periodically, the backup app running in VM2 will contact VM1 to pull the data from the regular volume. VM1 keeps track of the files being modified in this period T, and the corresponding encrypted markers and file IDs will be pulled by the backup app. The backup app will then decrypt all the markers using the private key, and identify those "important" files which have been changed. It then requests VM1 to send back those files. To save both communication and storage, the backup can be performed in an incremental manner. Specifically, only the difference (i.e., delta) between two file versions will be transmitted and stored.

The detailed backup process is shown in Algorithm 2. In this algorithm, the backup app first pulls the attributes from VM1 for those files which have been changed in this period (i.e., PullAttributes). For each file, the backup app obtains the "marker" attribute (i.e., GetFileAttribute), decodes it using base64 (i.e., Base64Decode), which is then decrypted using the private key sk. By removing the file ID, we can simply obtain the marker, which determines whether we should back up the corresponding file or not.

Algorithm 2 Back up data

```
1: procedure BackUp(Host, sk)
2:     S ←PullAttributes(Host)
3:     for ∀s ∈ S do
4:         attribute ← GetFileAttribute(s)
5:         ciphertext ← Base64Decode(attribute)
6:         plaintext ← ADec_sk(ciphertext)
7:         obtain marker from plaintext
8:         if marker then
9:             pull this file from regular volume using delta
   encoding
10:    return true
```

Data recovery. Once the detector component has detected and blocked a piece of ransomware, it will inform the recovery app running in VM2. This process usually requires the involvement of the device owner. Specifically, the data owner needs to provide the credential (e.g., a secret password) in VM1 which is used to pass the authentication of VM2.

After the VM1 successfully passes the authentication, the recovery app will check the backup volume to see whether a corrupted file has been backed up previously. If a backup copy is found, the recovery app will reconstruct this copy (this usually requires starting from the initial file version, and applying all the subsequent deltas [18]), and send it back to VM1. Otherwise, this corrupted file does not belong to the "important" files being protected, and no recovery action will be performed. After the recovery is done, the credential should be completely removed from VM1.

VI. Analysis and Discussion

A. Security Analysis for RDS3

RDS3 defends against ransomware by periodically backing up data. However, since we cannot predict when ransomware is present to corrupt the data, it will be possible that ransomware will corrupt the data which have not been backed up. We analyze this security leakage in the following.

Let t_1 be the point of time ransomware is present and starts to encrypt the data, and t_2 be the point of time the

ransomware is detected and blocked. Let t' be the point of time before t_1 upon the latest back up process is performed. Recall that T is the time interval for invoking a backup process. We have two cases:

1) The "important" data created at or before t' are always recoverable since they have already been backed up; 2) The "important data" created between t' and t_2 may be corrupted by ransomware.

For case 2), we first quantify this vulnerable period (we call it *vulnerable window* here). The vulnerable window can be computed as: $t_2 - t'$. As t' is the point of time before t_1 when the latest back up process was performed, we have: $t' + T > t_1$, i.e., $t' > t_1 - T$.
Thus, $t_2 - t' < t_2 - (t_1 - T) = (t_2 - t_1) + T$.
If there exists an effective detector which can detect ransomware within T, then $t_2 - t_1 < T$.
In this case, $t_2 - t' < (t_2 - t_1) + T < T + T = 2 \cdot T$. If T is chosen as a few minutes, then this vulnerable window will be also a few minutes. We further understand how the ransomware will affect the "important" data during the aforementioned vulnerable window. Let δ be the amount of "important" data being generated in this vulnerable window, and G be the total amount of data in the regular volume. Assume the ransomware can encrypt x bits of data per second, then:
The amount of data being encrypted by the ransomware is: $(t_2 - t_1) \cdot x$.
As the ransomware has no knowledge of which data are "important," it can only corrupt data in a random (uniformly random) manner. Thus, the amount of "important" data encrypted by ransomware will be:
$(t_2 - t_1) \cdot x \cdot \frac{\delta}{G}$.
For example, if the detector can detect ransomware in 120 seconds, and ransomware can encrypt 100MB/s; the amount of "important" data generated during the vulnerable window is 100MB; the total amount of data in the regular volume is 500GB, then the amount of "important" data encrypted by ransomware will be 2.4MB, which is 2.4% of the entire newly generated "important" data.

B. Discussion

Isolation provided by a virtual machine monitor. RDS3 relies on the isolation provided by a virtual machine monitor to prevent the ransomware from accessing the backup volume. Although a few existing works [35, 36] were investigating the vulnerabilities of virtualization, most of them are focusing on stealing information from a co-resident virtual machine, which are more concerned with attacking read operations. Our objective in RDS3, however, is to prevent the attacker from writing the co-resident virtual machine, which is more concerned with write operations.

About the "spare space" assumption. RDS3 relies on the assumption that the computing device is equipped with a certain amount of spare space (see Section III), which can be utilized to create the backup volume. To justify this assumption in practice, we conducted a survey in our institute. We selected a group of 104 students, and obtained the storage usage in their personal computers. The statistical results

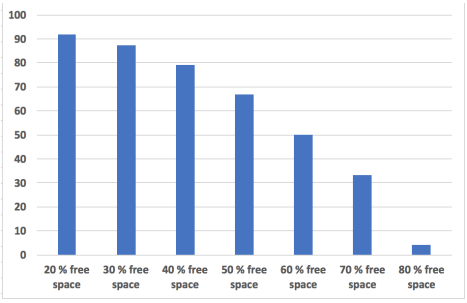

Fig. 3: Disk usage statistics in our survey. The bar for "x% free space" shows the percentage of subjects who have at least x% free space

are shown in Figure 3. The results show that most of the computing devices (80%-90%) have more than 20% spare space. This confirms the practicality of RDS3.

Computing delta. During the back up process, the backup app needs to compute the difference (i.e., delta) between the file version stored in the backup volume and that stored in the regular volume. Rolling checksum [34] can be utilized to efficiently identify the difference of the two files stored in different machines. In addition, to reduce the storage cost for the backup data, we can periodically perform an in-place reconstruction on the backup volume, to obtain a new starting-version for each file.

Operation suggestion. To ensure the ransomware will not hack into the tiny OS for the backup volume, we recommend the device owner should not enter the tiny OS. By default, the device should directly boot into the user OS for daily use. The tiny OS can be activated by the virtual machine monitor whenever a backup process or data recovery process is needed.

Mitigating the "marking" attack. Marking a file as "important" only relies on the public key, which is also known to the ransomware who can obtain root privilege. Therefore, the ransomware may disturb the system by marking all the files as "important." We claim this is not a major issue, because: first, the recovery component of RDS3 is actually a version control system which uses an incremental backup technique. The version control system can ensure that no data will be lost. Second, RDS3 also incorporates a detector component which can detect and block ransomware within a reasonable amount of time. Thus, to perform the aforementioned attack, the ransomware needs to mark a large number of files as "important" in a short time, and such abnormal behavior can be easily detected by the detector component running in VM1.

VII. Implementation and Evaluation

A. Implementation

We implemented a prototype for RDS3 by tailoring Linux as the operating system. We implemented various components in RDS3: *initialization*, *marker generation*, *back up data*, and *data recovery* in c. The implementation details are elaborated below.

TABLE I: Throughput of RDS3 in VM1 under different VMMs

Machine	SREAD	RANREAD	SWRITE	RANWRITE
no RDS3	53.9MB/s	815KB/s	146MB/s	749KB/s
XEN	31.6MB/s	806KB/s	97.2MB/s	732KB/s
VirtualBox	22.7MB/s	712KB/s	50.9MB/s	756KB/s

Initialization. The asymmetric encryption is instantiated using RSA with 1024-bit key size, i.e. $\lambda = 1024$. The key pair (pk, sk) is generated using OpenSSL cryptographic library.

Marker generation. Marker generation is performed in VM1. We use '1' and '0' as the marker to distinguish "important" and "not important" file, respectively. We concatenate marker, file ID, and nonce (randomly generated during run time), and encrypt them using RSA asymmetric encryption. The resulting cipher is further encoded using base64 encoding[2], generating the corresponding file attribute. The attribute is further affiliated with the corresponding file by running setfattr [9]. By default, when a new file is created, it will be set with an attribute being computed using marker "0". The inotify API is used to monitor the changes of the file system which can detect a file creation event.

Back up data. We implement a backup app by leveraging rsync. Periodically, the backup app running in VM2 will read from the regular volume (over ssh) and decrypt the marker attributes of all the files using the private key. It then only backs up those files marked as "important" in an incremental manner using rsync (over ssh). If a user marks a large number of files as "important," there may be insufficient storage space in the backup volume. Therefore, the backup app also incorporates quota and notification functionality to inform the user about the space insufficiency.

Data recovery. We also implement a recovery app which runs in VM2 and can be used by the victim to restore the data corrupted by the ransomware, using the data stored in the backup volume. Those corrupted files can be reconstructed in VM2 and then pushed back to the regular volume (over ssh), over-writing their corrupted versions.

B. Experimental Evaluation

We set up a test-bed for our experimental evaluation, using the following system configurations: Our host machine was equipped with Intel(R) Xeon(R) CPU X5550 @ 2.67GHz, 6GB RAM, and 1.8 TB disk space. Two VMMs (XEN and VirtualBox) were installed in the host machine running Ubuntu 16.04.2 LTS. Two virtual machines *VM1* and *VM2* were created under each VMM, respectively, each was assigned 1 core CPU, 1GB RAM, and 100GB hard disk space. We used a benchmarking tool fio [1] to evaluate the throughput. We compared RDS3 using different VMMs. We also collected the throughput of the host physical machine without running RDS3.

[2]Base64 encoding is used to facilitate storing and transmission of this attribute.

We evaluated the throughput in terms of sequential read (i.e., SREAD), random read (i.e., RANREAD), sequential write (i.e., SWRITE), and random write (i.e., RANWRITE). The experimental results are shown in Table I. We observed that the additional overhead introduced by RDS3 varies 41%-58% for sequential read, and 30%-65% for sequential write. The additional overhead mainly comes from the virtualization for isolation and computation required for backing up data. We did not observe significant additional overhead for random read and random write. The potential reason is, due to the internal implementation of virtualization, the random read/write may be not exactly equivalent to that in a physical machine. Specifically, the VMM may optimize the random seek issued from the upper layer. In addition, we observed that RDS3 running in XEN achieved better performance compared to running in VirtualBox. This actually indicates that XEN is optimized more in performance compared to VirtualBox.

VIII. Related Work

Ransomware mitigation in PC computers. Kharraz et al. [26] presented results of a long-term study of ransomware attacks that have been observed in the wild between 2006 and 2014. They showed how ransomware attacks have been evolved by analyzing more than 1,300 samples belonging to 15 different categories of ransomware. They also showed that ransomware can be detected by monitoring the activities in file systems. In addition, Kharaz et al. [25] created an artificial user environment and detected ransomware when it tried to interact with the environment. Another file system-based ransomware detection was performed in [32]. ShieldFS [21] created a protective wrapper around the windows operating system that were immune to Ransomware. It monitors the low-level file system activity and updates an adaptive model. Whenever a process violates that model, it marks that process as malicious and transparently recovers all the original files. PayBreak [27] tried to recover the data corrupted by ransomware by extracting the encryption key. However, such a solution can only work when the ransomware utilizes symmetric encryption or hybrid encryption.

Modern techniques such as Software Defined Networking (SDN) were used to detect and mitigate Ransomware attacks [16, 17]. Other techniques such as honeypot-based ransomware detection [29] were also explored. Ahmadian et al. [15] detected the Ransomware by monitoring the connection between the victims computer and the CC server.

IX. Conclusion

In this work, we propose RDS3, a ransomware defense approach which focuses on restoring the data corrupted by ransomware. By isolating spare space containing backup data from the regular volume, and making the spare space inaccessible for the ransomware, we make it possible to recover the encrypted data, regardless of what privilege the ransomware can obtain. Security analysis and experimental evaluation show that RDS3 is able to mitigate ransomware attacks with an acceptable performance overhead.

ACKNOWLEDGMENT

This work was supported by the Cluster to Advance cyber Security & Testing (CAST), FedEx Institute of Technology in the University of Memphis.

REFERENCES

[1] fio: Linux man page. https://linux.die.net/man/1/fio.

[2] Hyper-v. https://technet.microsoft.com/en-us/library/mt169373(v=ws.11).aspx.

[3] inotify - monitoring filesystem events. http://man7.org/linux/man\-pages/man7/inotify.7.html.

[4] Internet security threat report. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf. (2016).

[5] Ransomware a growing enterprise threat. https://go.crowdstrike.com/rs/281-OBQ-266/images/WhitepaperRansomware.pdf.

[6] Ransomware attack hit san francisco train system. https://www.usatoday.com/story/tech/news/2016/11/28/san-francisco-metro-hack-meant-free-rides-saturday/94545998/. (2016).

[7] Ransomware attackers collect ransom from kansas hospital. http://www.healthcareitnews.com/news/kansas-hospital-hitransomware-pays-then-attackers-\demand-second-ransom. Jan 9 2017.

[8] Ransomware: Now a billion dollar a year crime and growing. http://www.nbcnews.com/tech/security/ransomware-now-billiondollar-year-crime-growing-n704646. Jan 9 2017.

[9] setfattr: Linux man page. https://linux.die.net/man/1/setfattr.

[10] Stealth hooking : Another way to subvert the windows kernel. http://phrack.org/issues/65/4.html.

[11] University pays 16,000 to stop ransomware attack. http://fortune.com/2016/06/08/university-ransomware. Jan 9 2017.

[12] vsphere and vsphere with operations management. http://www.vmware.com/products/vsphere.html.

[13] wannacry. http://money.cnn.com/2017/05/12/technology/ransomware-attack-nsa-microsoft/.

[14] Xenserver - server virtualization and consolidation - citrix. https://www.citrix.com/products/xenserver/.

[15] M. M. Ahmadian, H. R. Shahriari, and S. M. Ghaffarian. Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares. In *Information Security and Cryptology (ISCISC), 2015 12th International Iranian Society of Cryptology Conference on*, pages 79–84. IEEE, 2015.

[16] K. Cabaj, M. Gregorczyk, and W. Mazurczyk. Software-defined networking-based crypto ransomware detection using http traffic characteristics. *arXiv preprint arXiv:1611.08294*, 2016.

[17] K. Cabaj and W. Mazurczyk. Using software-defined networking for ransomware mitigation: the case of cryptowall. *IEEE Network*, 30(6):14–20, 2016.

[18] B. Chen and R. Curtmola. Auditable version control systems. In *NDSS*, 2014.

[19] R. Chong. Locky ransomware distributed via docm attachments in latest email campaigns. *FireEye (17 Aug 2016) Accessed Sep*, 2016.

[20] L. Constantin. Cryptowall ransomware held over 600k computers hostage, encrypted 5 billion files. *IDG News Service*, 29, 2014.

[21] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi. Shieldfs: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 336–347. ACM, 2016.

[22] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, pages 171–172. IEEE, 2015.

[23] R. Hund, T. Holz, and F. C. Freiling. Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms. In *USENIX Security Symposium*, pages 383–398, 2009.

[24] K. Jarvis. Cryptolocker ransomware, 2013. *URL http://www.secureworks. com/cyber-threat-intelligence/threats/cryptolocker-ransomware/. R etrieved on April*, 21:30–31, 2014.

[25] 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)*, pages 757–772, 2016.

[26] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda. Cutting the gordian knot: a look under the hood of ransomware attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24. Springer, 2015.

[27] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele. Paybreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 599–611. ACM, 2017.

[28] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[29] C. Moore. Detecting ransomware with honeypot techniques. In *Cybersecurity and Cyberforensics Conference (CCC), 2016*, pages 77–81. IEEE, 2016.

[30] V. Oracle. Virtualbox user manual, 2011.

[31] M. H. U. Salvi and M. R. V. Kerkar. Ransomware: A cyber extortion. *Asian Journal of Convergence in Technology*, 2016.

[32] N. Scaife, H. Carter, P. Traynor, and K. R. Butler. Cryptolock (and drop it): stopping ransomware attacks on user data. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 303–312. IEEE, 2016.

[33] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. *arXiv preprint arXiv:1609.03020*, 2016.

[34] A. Tridgell, P. Mackerras, et al. The rsync algorithm. 1996.

[35] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *USENIX Security symposium*, pages 159–173, 2012.

[36] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.