Hail to the Thief: Protecting Data from Mobile Ransomware with *ransomSafeDroid*

Sileshi Demesie Yalew[1,2], Gerald Q. Maguire Jr.[2], Seif Haridi[2], Miguel Correia[1]

[1]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
[2]School of Information and Communication Technology, KTH Royal Institute of Technology, Sweden
sdyalew@kth.se, maguire@kth.se, haridi@kth.se, miguel.p.correia@tecnico.ulisboa.pt

***Abstract*—The growing popularity of Android and the increasing amount of sensitive data stored in mobile devices have lead to the dissemination of Android ransomware. Ransomware is a class of malware that makes data inaccessible by blocking access to the device or, more frequently, by encrypting the data; to recover the data, the user has to pay a ransom to the attacker. A solution for this problem is to backup the data. Although backup tools are available for Android, these tools may be compromised or blocked by the ransomware itself.**

This paper presents the design and implementation of RANSOMSAFEDROID, a TrustZone based backup service for mobile devices. RANSOMSAFEDROID is protected from malware by leveraging the ARM TrustZone extension and running in the secure world. It does backup of files periodically to a secure local persistent partition and pushes these backups to external storage to protect them from ransomware. Initially, RANSOMSAFEDROID does a full backup of the device filesystem, then it does incremental backups that save the changes since the last backup. As a proof-of-concept, we implemented a RANSOMSAFEDROID prototype and provide a performance evaluation using an i.MX53 development board.

I. INTRODUCTION

Ransomware (such as WannaCry [1]) has been appearing a lot in the news. Ransomware is malware that prevents access to data in a computer, either by locking the system's screen or, more frequently, by encrypting files, then demanding a ransom from the victim to provide back that access [2]. The ransom is typically paid in a cryptocurrency (e.g., Bitcoin or Monero).

Ransomware has begun to attack mobile devices running the Android operating system (OS) similarly to what happened earlier with PCs [3]–[5]. Android ransomware can be divided in two classes: lock-screen ransomware and crypto-ransomware. *Lock-screen ransomware* (such as *Android.Lockdroid.E*) blocks user interaction with the device, for example by leveraging the SYSTEM_ALERT_WINDOW Android permission to lock the screen. An application (app) that has access to this permission can create system-type windows and display them on top of every other app or window, making it impossible to use the device. Before Android 6.0 Marshmallow, this permission was granted automatically to any app from Google Play Store that requested it, facilitating this form of attack [6]. Nevertheless, this class of ransomware seems to be quite specific, as it depends on particular design vulnerabilities.

978-1-5386-1465-5/17/$31.00 ©2017 IEEE

The most common form of ransomware is *crypto-ransomware* (such as *Trojan-Ransom.AndroidOS.Small* or WannaCry for PCs), which encrypts files in the victim's device and demands payment to provide the decryption key. Interestingly, paying the ransom may not return the files, as sometimes the attacker does not provide the key. Moreover files may be corrupted and cannot be decrypted. Making regular backups of the files [7] is a common solution to this type of attack.

There are several tools to backup and restore data and apps in mobile devices. For example, *Samsung Smart Switch* [8] and *LG Bridge* [9] transfer data (e.g., documents, videos, pictures, and contacts) and apps from a mobile device to a PC and vice-versa. Moreover, mobile OSes (such as Android and iOS) provide utilities to easily make cloud backups. However, the existing backup tools are run in the same execution environment as the malware that infects the device, hence they may be compromised or blocked by the ransomware. In fact, malware is often able to disable anti-malware software and other security software [10], as shown by the recent case of HijackRAT [11]. From the attacker's viewpoint, an effective approach would be for the ransomware to disable the backup tool(s) for a period of time before it starts encrypting the files, so files that have been backed up are outdated. Furthermore, in some cases ransomware is able to encrypt or delete the backup files themselves, as happened with WannaCry itself, which deletes shadow copies of a particular volume [1]. Therefore, the backups themselves have to be protected. Unfortunately, this is something that does not happen for the online solutions just discussed.

TrustZone is a hardware security extension provided by recent ARM processors to enable trusted computing [12], [13]. The aim of TrustZone technology is to provide two execution environments: the *secure world*, a trusted execution environment where trusted code runs, and the *normal world*, where untrusted code, including the mobile OS (e.g., Android) and the mobile apps, is executed. The physical core of the processor is divided into two virtual cores, corresponding to the two environments. Memory space, peripherals, interrupts, and other resource can be assigned to the secure world, hence they are isolated from the normal world. Moreover, code running in the secure world can access the resources of the normal world.

This paper describes the design and implementation of RANSOMSAFEDROID, a TrustZone-based backup service for protecting data in mobile devices from crypto-ransomware. We leverage TrustZone to protect RANSOMSAFEDROID from malware running in the normal world. Because RANSOMSAFEDROID runs in the secure world, it is secure despite the normal world, including the mobile OS, being compromised. RANSOMSAFEDROID is able to make a periodical backup of files in the normal world to a local storage partition in the secure world, and to push these backups to external storage (e.g, a cloud service). It offers fast incremental backups which capture only the changes made to files since the last backup and efficient storage compression and deduplication to remove redundant data, thus saving backup storage space.

To the best of our knowledge, in the existing TrustZone literature it is always the normal world that initiates a call to a trusted service in the secure world, using the `smc` instruction [14]–[20]. However, such a mechanism might be blocked by the ransomware, hence we have to follow a different approach and make the secure world active, instead of passive. Specifically, RANSOMSAFEDROID is scheduled to run automatically at specific times, without the intervention of the normal world. For this to happen, a hardware timer is used to generate a periodic interrupt which forces the execution of RANSOMSAFEDROID in the secure world. This means that ransomware running in the normal world cannot stop it.

The main contributions of this paper are: (1) the design of RANSOMSAFEDROID, a backup service for mobile devices that is protected using the TrustZone extension; (2) the design of an active secure world to periodically run RANSOMSAFEDROID by using a hardware timer, without the possibility of the normal world stopping it; (3) an implementation of RANSOMSAFEDROID in hardware, on the NXP Semiconductors i.MX53 Quick Start Board (QSB); and (4) an experimental evaluation of RANSOMSAFEDROID.

II. Background

This section provides background information on the main technologies underlying RANSOMSAFEDROID.

A. Android Storage Architecture

Mobile devices such as smartphones normally have three types of read/write memory: volatile RAM, non-volatile solid-state *internal storage* (e.g., eMMC), and an optional non-volatile solid-state *SD card*. Android apps can store persistent configuration files and data in the device's internal storage and in the SD card. Android uses different filesystem partitions to organize files and folders on the storage device. There are typically six partitions for the internal memory plus one partition on the SD card, i.e., bootloader (`/boot`), kernel (`/system`), device recovery (`/recovery`), user data (`/data`), data/component cache (`/cache`), miscellaneous settings (`/misc`), and SD card (`/sdcard`).

Android used the YAFFS2 [21] filesystem for the various internal partitions including `/system` and `/data`. YAFFS2 is lightweight and optimized for NAND internal storage. Recently, Android transitioned to *ext4* as the default file system for these partitions [22]. For compatibility, Android also provides a filesystem to access a FAT32 formatted external storage, as this is a commonly used file system on SD cards.

B. File Backup Schemes

Backup refers to copying files to an alternative storage location, typically secondary or remote storage, in order to prevent data loss due to human action, hardware and software failures, lost/theft of the device, or accidents/disasters. There are three main backup techniques: full backup, incremental backup, and differential backup.

In a *full backup*, all the files that have been selected for backup are copied. This approach is efficient when the number and aggregate size of files is small, but otherwise it can be slow. Moreover, it can be expensive if performed over a cellular network.

Incremental and differential backups aim to solve the issues of delay and cost. Both require an initial full backup, then only those files that have been modified or created are copied when subsequent backups occur. The difference between these two is that an *incremental backup* considers all changes since the previous backup (full or incremental), while a *differential backup* includes all files that have been changed or added since the last full backup. These two schemes result in faster and smaller backups in comparison with full backup, as long as only a small percentage of files changes before each subsequent backup. The advantage of reducing the size of backups is not only less storage space is needed, but also less data that needs to be transferred over the network in the case of remote backups.

All of the above backup methods can be combined with other methods of reducing the backup storage space and network bandwidth, such as data compression [23] and deduplication [24], [25].

There is a vast range of *compression* techniques, but all are based on the idea of removing redundancy in order to store data in a more compact form. Compression techniques may be lossless (the original data can be fully recovered) or lossy (the original data cannot be entirely recovered).

Deduplication aims to remove redundancy with respect to files stored in the same data store. In file-level deduplication, a file is not copied if it already exists in the store. In block-level deduplication, files are broken in blocks of the same size; a block is not copied if it already exists in the store.

C. ARM TrustZone

ARM TrustZone [12] is a hardware security extension incorporated into recent ARM processor designs (e.g., Cortex A8, A9, and A15). The aim of this technology is to enable a device to offer both a feature-rich open operating environment and a robust security solution by partitioning the system-on-chip hardware and software into two execution environments or worlds: the *secure world* that runs trusted apps on top of a small trusted OS; and the *normal world* that runs untrusted apps on top of a rich OS such as Android. The secure world is logically separated from the normal world, but exists in the same physical CPU. A context switch between these two worlds is triggered by an interrupt, typically a software interrupt triggered by a call to the *secure monitor call* (`smc`) instruction. ARM TrustZone enables any part of the system to be assigned to the secure world. The normal world cannot access those system resources (e.g., memory space and peripherals) that are assigned to the secure world; while the secure world has access to the normal world's resources.

III. RANSOMSAFEDROID

This section presents the architecture and design of RANSOMSAFEDROID.

A. Threat Model and Assumptions

We assume a device with an ARM processor with the TrustZone extension. RANSOMSAFEDROID runs in the secure world, isolated from the normal world. In order to minimize the size of the trusted computing base (TCB) [26], i.e., the software in the secure world, we run a small custom kernel in that world, and do not include a network stack. We assume the secure world software, including RANSOMSAFEDROID, is verified as trusted. In contrast, we assume that the software running in the normal world, including the mobile OS (typically Android), may be malicious or compromised, e.g., by ransomware.

The device is configured with a hardware timer reserved for the secure world. While malware or attackers might be interested in disabling this timer, we assume they cannot access or compromise the resources assigned to the secure world. We also assume that, if the user wishes to initiate a restore of their files, the user interface for this operation is secured by using TrustZone (this has previously been addressed in [19]). In all cases we assume that the attacker does not have physical access to the device.

B. Architecture

The architecture of RANSOMSAFEDROID is shown in Figure 1. The normal world runs a mobile OS and apps, whereas the secure world runs RANSOMSAFEDROID on top of a small trusted OS that provides basic OS functions (e.g., process management, file access, and memory management).

The *secure storage* is a private persistent partition used for local backup storage. It is isolated in the secure world, i.e., it cannot be accessed by the normal world. This way, backups are protected from malware running in the normal world despite the mobile OS being compromised by ransomware.

RANSOMSAFEDROID has three software modules: local backup, external backup, and restore. The *local backup* module copies files in the normal world into the local secure storage, whereas the *external backup* module pushes backups stored in secure storage to an external device, a remote server, or cloud computing service (e.g., Amazon S3 or Google Drive). The *restore* module copies files from a backup (local or remote) into the normal world, removing the effects of ransomware attacks or other failures.

C. Active Secure World

To protect mobile devices from ransomware, not only must the backups be protected, but the adversary must be prevented from denying the execution of the backup/restore operations. Therefore, backups/restores cannot be initiated by a process in the normal world. This implies that the secure world has to be active, i.e., has to run RANSOMSAFEDROID without being called. As far as we are aware, this is the first such TrustZone use proposed in the literature.

In order to generate an interrupt to initiate backup/restore operations, RANSOMSAFEDROID configures a hardware timer to be accessible only by the secure world, preventing the normal world from disabling it. This timer will generate a periodic interrupt that triggers the execution of the secure world. In the discussion that follows we focus specifically on the local backup module as this is likely to be the most frequent operation.

The period T_{sw} (secure world execution period) is configurable. There is a tradeoff when setting T_{sw} as with a longer period the higher the probability some important file may not be backup when an attack happens; a shorter period increases the overhead, as the normal world execution is more frequently interrupted and the more likely it is that there are no changes in files to be processed.

The assigned hardware timer is not part of the ARM processor itself, but of the device, so it depends on the specific hardware used. Section IV explains the configuration for the i.MX53 board that was used to implement the prototype.

Next we present the main components of RANSOMSAFEDROID (as shown in Figure 1).

D. Local Backup Module

The local backup module copies files from the normal world and saves them to the secure storage in the secure world, for a later restore if the device is attacked and files are encrypted by ransomware.

This module creates in the secure storage (in the secure world) an index of the files and directories in the filesystem that is being backup (from the normal world). This index stores a list of files and their attributes, along with hashes over the files. This data is used to track the files in the index that have been updated since the last backup.

This module utilizes an incremental backup to save both space and time. Files are divided into blocks of data and each block is separately indexed by its hash/checksum. The scheme uses rolling checksums based on the rolling checksum algorithm from the *rsync* tool [27] to compare the file blocks, for determining the difference between two files and to save only the difference. This scheme allows identification of even small differences in large files, hence minimizing the number of blocks that have to be stored in the backup.

As explained in Section II-B, the first backup has to be a full backup. Doing a full backup of an Android filesystem takes some time, as it typically involves copying gigabytes of data. Moreover, mobile devices tend to be slower than PCs, with respect to processor speed, internal bandwidth, and memory

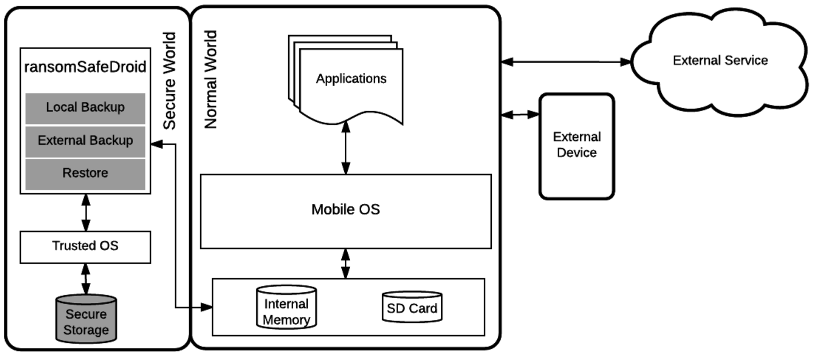

Fig. 1. Architecture of a mobile device running RANSOMSAFEDROID. The grey boxes are components of RANSOMSAFEDROID.

speed. However, the overhead of the full backup does not have an impact on usability, as it should be done before the device enters normal operation. For example, it can be done by the device manufacturer before it is sold to the end-user.

Moreover, the backup module uses both compression and deduplication. In terms of compression, the idea is simply to compress the files using zip-like lossless compression algorithms. In relation to deduplication, the idea is to use hashes of the file blocks to identify and remove redundant data across files and generations of backups. If the backup repository already contains a particular block of data, it will be re-used and the duplicated block of data will be replaced with a link. This feature can significantly reduce the amount of storage space needed for backups, which is important for mobile devices since they normally have limited storage space.

The secure storage should be large enough to contain the full backup and the increments, but it cannot have infinite space. To deal with this limitation, the major mechanism is the external backup (next section). Moreover, if the space available goes below a certain threshold, the user may be told explicitly to do an external backup, e.g., using an Android notification (that appears on the top of the screen).

E. External Backup Module

Local backups are effective against ransomware, but not against other more classical threats, such as the device being lost, stolen, or destroyed. Therefore, we extend the basic RANSOMSAFEDROID design with the ability to perform backups to external storage, e.g., to remote servers or to a remote cloud computing storage service.

As local backups are already compact, the remote servers simply maintain a copy of the local backups. A naive solution would be to simply copy all new and modified files to the remote storage whenever they appear in the (local) secure storage. However, this would considerably increase the overhead of doing the backups. Therefore, we use an opportunistic and cautionary approach: whenever a local backup module is executed, it measures the time it takes to run, i.e., t_{lb} (local backup time). Then, it passes control to the external backup module that copies files to remote storage during at most $t_{max} - t_{lb}$ units of time (if positive). The interval of

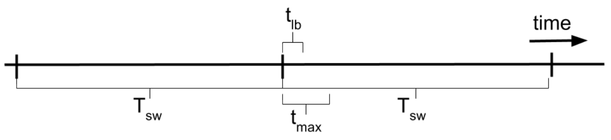

Fig. 2. Backup period and other relevant times.

time t_{max} (maximum time) is a configurable parameter that indicates the maximum time the backup process should take, i.e., that the secure world should run in every period T_{sw}. It is important to limit this time because ARM CPUs do not do time sharing between the two worlds, hence when the secure world is running the normal world is blocked (and vice-versa). These times are represented in Figure 2.

External backups can be done in two ways. The first is by connecting a storage device (e.g., an external disk or an SD card) directly to the mobile device. This solution requires allocating the I/O devices needed for this purpose to the secure world in order to prevent malware in the normal world from corrupting such backups. Second, the backup can be done via a network. As we explained earlier, the secure world does not contain a network stack in order to reduce the size of the TCB. Therefore, backups via the network have to be done with the assistance of a gateway app in the normal world (not shown in the figure, as this is optional). Additionally, end-to-end encryption and message authentication codes for security must be used (similarly to the solution in [16]). As a result, malware in the normal world may be able to prevent this communication, but cannot corrupt it. Nevertheless, recall that the purpose of remote backups is not to protect against ransomware but rather to protect against other forms of threats.

F. Restore Module and App

This module has the role of restoring files when needed. It has the ability to restore files from local or remote storage.

The restore operation is normally started using a *restore app* that is executed in the normal world and allows the user to define how the restore is done. This app calls the restore module in the secure world using the `smc` instruction. The user has to provide authentication credentials in order to prove to RANSOMSAFEDROID that they are a legitimate user, rather than an adversary. The authenticity of the app may be checked using TruApp, a TrustZone-based service to check app authenticity and integrity [20].

When a restore is done after a ransomware attack, most likely the most recent backups will consist of encrypted files. Incremental backups allow dealing with this problem by not recovering the increments that correspond to encrypted files. The detection of which increments correspond to encrypted files can be done using a few heuristics. First, when a whole filesystem is encrypted by ransomware, there must be an easy to observe peak in the number of files to be backup, from a few to many. Second, the restore app may include a scheme to detect which files are encrypted, e.g., using entropy analysis [28].

The restore operation tends to be quite slow, because it has first to copy the initial full backup, then all the changes that were made. However, while the backups are executed during normal operation, restore is an exceptional operation, executed only when a problem occurs (e.g., a ransomware attack). Therefore, the time to restore is not considered critical for the usability of RANSOMSAFEDROID.

IV. IMPLEMENTATION

In this section, we discuss the implementation of the RANSOMSAFEDROID prototype on an i.MX53 QSB board. The board is equipped with a Cortex A8 single-core 1 GHz processor and 1 GB RAM (DDR memory). We choose this board because most commercial TrustZone enabled smartphones are locked and do not allow programming the secure world.

In order to minimize the size of the TCB in the secure world, we setup a small trusted OS based on a custom kernel (base-hw) provided by the Genode labs [29] for our board. In the normal world, we installed a version of Android for the i.MX53 series from Adeneo/Freescale [30]. The kernel is patched by the Genode project to be executed in the normal world. The DDR RAM is partitioned into secure world and normal world address ranges.

A TrustZone monitor called *tz_vmm* is implemented in the secure world as a user level program that runs on top of the Genode kernel. In the secure world, we implemented RANSOMSAFEDROID based on *tz_vmm*. We configured the Android filesystem partitions to be accessed by the secure world, so RANSOMSAFEDROID modules can access and copy files in the normal world.

A secure storage space has to be reserved in the secure world, to be accessible exclusively by the secure world components, in order to store the local backups (secure storage in Figure 1). We use the Genode partition manager (*part_blk*) for this purpose. It makes each partition on a SD card available as a block session. This allows the partitions to be addressable as separate block sessions and makes it is easy to grant or deny access to them. In our prototype, we used this scheme to assign an SD card partition to the secure world. In the current prototype, we allocated 50 GB of secure storage space from a total of 128 GB space available on a microSDHC UHS-I Class U3 card. This card supports speeds up to 90 MB/s for reads and 80 MB/s for writes.

In the prototype, the local backup module is based on *bup* [31], an open source backup tool, selected after comparing the performance of five similar software packages. *bup* is an efficient file backup software based on the *git* packfile format. *bup* offers incremental backups, compression using the *zlib* library (the default in the *git* packfile format), and storage deduplication, thus providing backup time and space savings. This tool has several modules, including the *index* module to create an index of files, and the *save* module that creates a new backup. Unfortunately, *bup* is written in Python, which requires adding the Python runtime to the TCB. Nevertheless, this is just a prototype and for production purposes *bup* should be replaced by code written in C/C++ or another language that does not require a runtime engine. Moreover, the *bup* website acknowledges that "Writing more parts in C might help with the speed" [31].

In order to make the secure world active, we used the *enhanced periodic interrupt timer* (EPIT) available on the i.MX53 board [32]. EPIT is a 32-bit set-and-forget timer. A driver for the EPIT timer has been implemented by the Genode project in the *base-hw* kernel running in the secure world, allowing the configurable periodic execution of the secure world. In i.MX53, groups of I/O devices are assigned to one of the worlds using configuration bits of the *central security unit* (CSU), similarly to what happens with the protection controller used on ARM's versatile express platform [29]. In our configuration, a group containing the EPIT is assigned to the secure world. This is done essentially by modifying a Genode configuration file (`csu.h`), where hardware assigned to the secure world is tagged `secure`, whereas hardware assigned to the normal world is tagged `unsecure`.

We focused most of our implementation effort in the mechanisms just described, which are the most relevant for the normal operation of RANSOMSAFEDROID. On the contrary, the implementations of the external backup module, the restore module and the restore app are not finished yet.

V. EXPERIMENTAL EVALUATION

In this section, we evaluate RANSOMSAFEDROID in terms of backup latency and storage efficiency by considering three different circumstances: initial backup, runtime backups, and null backups. We consider only local backups, because this is the main functionality of RANSOMSAFEDROID and the performance of remote backups depends strongly on the t_{max} and t_{lb} parameters as well as the properties of the communication with the remote store. As explained in Section III-F, we also do not evaluate the restore time because this is considered to be an exceptional operation that does not impact usability.

A. Initial Backup

We conducted experiments to understand the cost of an initial (full) backup in terms of time and storage space. To make these experiments realistic, we used a real Android filesystem with a 1.1 GB storage size. We ran RANSOMSAFEDROID

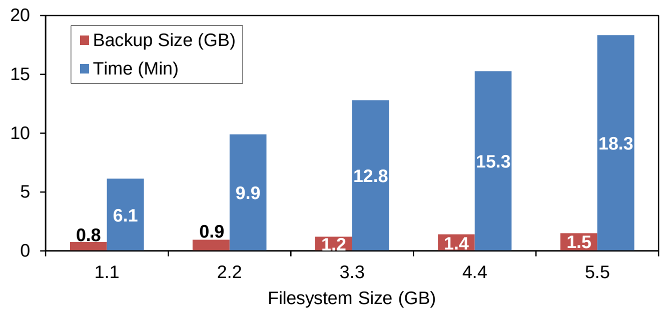

Fig. 3. Initial/full backup with different Android filesystem sizes.

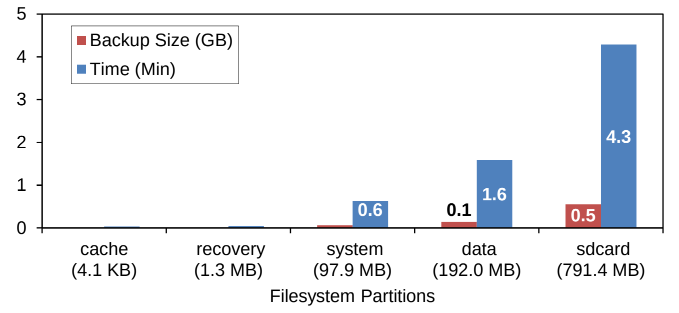

Fig. 4. Initial/full backup of Android filesystem partitions (1.1 GB filesystem).

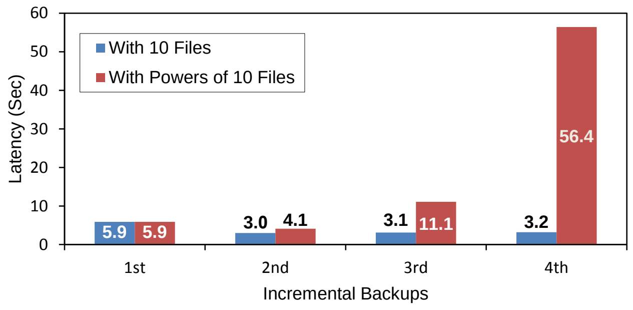

Fig. 5. Latency to make incremental backups when the filesystem is increased by 1% with 10 equally sized files and with powers of 10 files in each backup.

to perform a full backup of all files in this filesystem, and measured elapsed time and size of the backup created in the secure storage (in the secure world). Additionally, we repeated this experiment for different filesystem sizes: 2.2 GB, 3.3 GB, 4.4 GB, and 5.5 GB. For this purpose, we prepared different filesystems by adding new files to the real Android filesystem in multiples of 1.1 GB. We use the *genbackupdata* tool [33], which generates test data sets for performance evaluation of backup tools. Each data set consists of files and directories. Files can be either text files or binary files. In these tests we used 10 equally sized files and set equal percentages of text files and binary files.

The results of these experiments are shown in Figure 3. A first conclusion is that the full backup takes a considerable time, e.g., 6.1 minutes for the smaller filesystem and 18.3 for the largest. This may seem to be deterrent for the use of this service; but, as explained in Section III-D, the full backup can be done before the device is in normal use, even before it is sold to the end-user. A second conclusion is that the time grows approximately linearly with the size of the filesystem. Finally, it is noteworthy that the read/write speeds of the SD card are not the performance bottleneck, as the time to read and write 5.5 GB from the card at the (maximum) speeds of 90 and 80 MB/s is 2.16 minutes, which is far less than the observed 18.3 minutes. The bottleneck is the CPU speed, as the CPU used on the i.MX53 has 1 GHz clock speed.

Next, we did an experiment to show that it is possible to reduce the time of the full backup by doing it in steps. For that purpose, we evaluated the performance of a full backup of each partition or sub-tree in the filesystem. The 1.1 GB Android filesystem in the current prototype has 5 partitions: `cache`, `recovery`, `system`, `data`, and `sdcard`. For each partition, we measured the total time to complete a full backup of the partition and the storage space used by the backup data. We show these results in Figure 4. It is possible to observe that the total time for the full backup is greater than the 6.1 minutes observed in Figure 3. However, the times to backup the individual partitions is smaller, so they might be easier to do during idle times, if that was required.

B. Runtime Backups

The overhead of runtime backups depends on two factors: the period of the incremental backups (T_{sw}) and their size. As the period is configurable, we did not take it into account in the evaluation. However, one might assume that this could occur when the device is otherwise unused and recharging (perhaps each night).

We considered the case of the size of a filesystem increasing by 1% in every step to emulate incremental changes. For this purpose, we took as a basis the full backup of the 1.1 GB filesystem, then we created new files using the *genbackupdata* tool as described earlier.

We performed two experiments by considering the number of new files created in each step after the initial full backup of the Android filesystem. In the first experiment, we increased the size of the filesystem by 1% with 10 equally sized files in each incremental backup; in the second experiment we also incremented the size of the filesystem with different numbers of equally sized files in powers of 10 (e.g., 10, 100, 1000 and 10000 files) in each step. Note that the total number of bytes that are backed up in each incremental backup are the same in both experiments: 1% of the size of the filesystem.

We measured the latency and backup size for the two experiments. We show the results in Figures 5 and 6. A first conclusion is that the times are much smaller than those observed for the full backup, as expected as the amount of the data is much smaller (times are in seconds, no longer in minutes). A second conclusion is that backups of more files take more time, e.g., the backup of 11 MB with 10 files and 100 files – second backup – takes respectively 3.0 and 4.1 seconds.

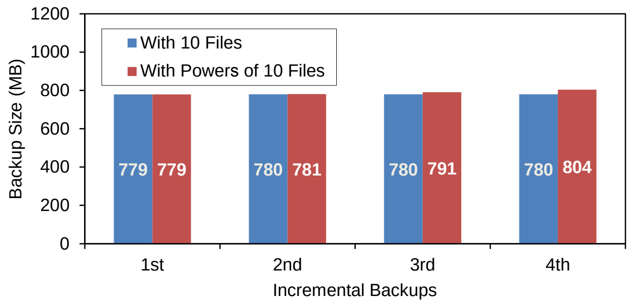

Fig. 6. Storage space usage to make incremental backups when the filesystem is increased by 1% with 10 equally sized files and with powers of 10 files in each backup.

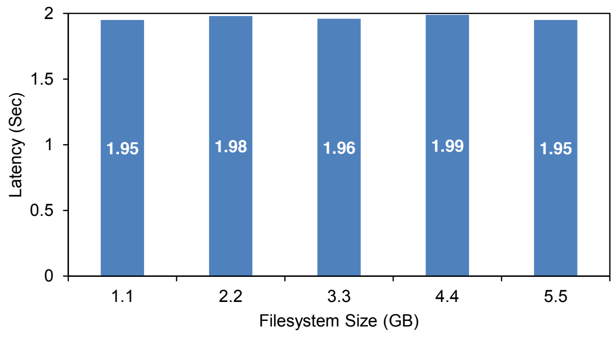

Fig. 7. Latency of null backups, i.e., incremental backups when the filesystem was not modified.

C. Null Backups

Finally, we also measured the time it takes to run RANSOMSAFEDROID and do an incremental backup when the filesystem did not change, i.e., the time spent with null backups. This may happen quite often, depending on the periodicity with which RANSOMSAFEDROID is executed (i.e., depending on T_{sw}), hence it is important to understand how much time null backups take.

As before, we considered different sized filesystems (i.e., 1.1 GB, 2.2 GB, 3.3 GB, 4.4 GB and 5.5 GB) and configured RANSOMSAFEDROID to periodically perform 10 null incremental backups for each filesystem size. We show the average of these 10 results for each different size of filesystem in Figure 7. We observe that the time needed to perform a null backup for all filesystem sizes is approximately 2 seconds on our board. This is the time necessary for RANSOMSAFEDROID to check if any change has been made since the last backup. The time may be even longer if the numbers and size of files in the filesystem are higher. Interestingly, the time seems to be independent of the size of the filesystem analyzed.

We studied if it is possible to reduce this delay of 2 seconds by implementing an alternative mechanism to check if files were modified. For that purpose, we implemented a small script using the *find* command. We obtained a delay of 20 milliseconds, with a standard deviation of 6 milliseconds, which are much better values. This supports our claim that it is possible to improve performance with a more efficient implementation of the backup code, e.g., written in C/C++.

Notice that these times are measured between the moment the secure world starts to run until it returns, so they do not include the time for context switching between the two worlds. It is not possible to measure this latter time using the interrupt caused by the timer, so we created a routine to trigger the secure world by calling the `smc` instruction in the normal world, then returning immediately. This time is measured in the normal world just before and after the trigger. We obtained an average time of 45 microseconds (averaged over 1000 repetitions).

VI. RELATED WORK

There is a large variety of backup software for keeping copies of all data (photos, videos, music files, and contacts) and apps installed on smartphones and tablets. Some backup apps, e.g., *Samsung Smart Switch* [8] and *LG Bridge* [9], transfer data and apps to a PC using a USB cable and vice-versa. Mobile OSes such as Android and iOS also provide utilities that facilitate backing up of data to a remote storage cloud. Furthermore, Android also provides the *Auto Backup API* for developers to add backup functionality to their apps [34]. This API allows apps to backup data by uploading it to the user's Google Drive account, where it is protected by the user's Google account credentials. Nevertheless, all these backup schemes are exposed to malware as they run inside an untrusted execution environment where other applications and the mobile OS run [10], [11], [35]. Their security is based on a set of assumptions that are often broken: no permissions should be granted to download apps, no vulnerabilities should exist in certain apps, no vulnerabilities exist in the mobile OS, etc. In contrast, we use ARM TrustZone to protect our backup mechanism, based on the isolation it provides.

TrustZone has been utilized to provide a large set of security services [14], [16], [18]. For instance, T2Droid [17] enables secure dynamic analysis of Android apps by leveraging TrustZone. The Trusted Language Runtime (TLR) [14] provides a framework to separate application security logic, called a trustlet, from the rest of the application and runs it in the ARM TrustZone secure world. Several works on secure storage have proposed mechanisms to protect sensitive data such as private keys, that are only accessible to small security critical programs using the TrustZone [18], [19]. However, none of these works provides a backup solution or focuses on protection from ransomware, unlike our work. Moreover, all of these programs are designed such that it is the normal world that calls the secure world. In contrast, RANSOMSAFEDROID leverages an active secure world by using a hardware timer to initiate the secure world without intervention of the normal world.

Currently, most backup apps place the backed up files in rented remote servers or cloud storage, which do not belong to the owner of the files. This may lead to personal or confidential data being accessed without the consent of the owner [36], [37]. However, RANSOMSAFEDROID use a secure storage partition in the user's device, isolated from the untrusted execution environment, to store backups, thereby protecting unauthorized access to backup data. Additionally, data to be stored in a remote data store could be encrypted using a key stored only in the trusted local storage, thus protecting even externally stored backups.

File backup is a very old topic [7], [38], [39], but most of the emphasis of such earlier work was on backup speed and space efficiency. Our work, on the contrary, targets recent mobile devices that have particular characteristics, and focuses on protecting the backup mechanism using a modern hardware security extension (TrustZone).

VII. Conclusion

As mobile devices such as smartphones increase in popularity, so does the amount of sensitive data that people store on these devices. Seeing this as an opportunity, ransomware has begun to attack Android phones, mainly by encrypting files on these devices. Having a regular backup of files using backup applications is a common solution to the problem. However, the current backup systems are designed to run in the same execution environment as malware, hence this malware is able to block or disable these mechanisms. This paper presents the design, implementation, and experimental evaluation of ransomSafeDroid, a backup system that is protected from malware by leveraging ARM TrustZone and allows backing up files in a persistent local secure storage.

Acknowledgements This work was supported by the European Commission through the Erasmus Mundus Doctorate Programme under Grant Agreement No. 2012-0030 (EMJD-DC) and project H2020-653884 (SafeCloud), and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

[1] msft-mmpc, "Wannacrypt ransomware worm targets out-of-date systems," https://blogs.technet.microsoft.com/mmpc/2017/05/12/wannacrypt-ransomware-worm-targets-out-of-date-systems/, May 2017.

[2] 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*, 2015, pp. 3–24.

[3] N. Andronio, S. Zanero, and F. Maggi, "Heldroid: Dissecting and detecting mobile ransomware," in *International Workshop on Recent Advances in Intrusion Detection*, 2015, pp. 382–404.

[4] Kaspersky, "Mobile malware evolution 2016," 2016.

[5] D. Goodin, "Ransomware app hosted in Google Play infects unsuspecting Android user," Jan. 2017, https://arstechnica.com/information-technology/2017/01/ransomware-app-hosted-in-google-play-infects-unsuspecting-android-user/.

[6] D. Venkatesa, "Android Marshmallow will not go soft on mobile ransomware," https://www.symantec.com/connect/blogs/android-marshmallow-will-not-go-soft-mobile-ransomware, Sep. 2015.

[7] A. Chervenak, V. Vellanki, and Z. Kurmas, "Protecting file systems: A survey of backup techniques," in *Joint NASA and IEEE Mass Storage Conference*, vol. 99, 1998.

[8] "Samsung smart switch," http://www.samsung.com/nz/support/smartswitch/.

[9] "LG Bridge," http://www.lg.com/us/support/product-help/CT10000026-1438110404543-preinstall-apps.

[10] N. Leavitt, "Mobile phones: the next frontier for hackers?" *IEEE Computer*, vol. 38, no. 4, pp. 20–23, 2005.

[11] A. Greenberg, "Sneaky Android RAT disables required anti-virus apps to steal banking info," Jul. 2014, SC Magazine, https://www.scmagazine.com/sneaky-android-rat-disables-required-anti-virus-apps-to-steal-banking-info/article/538770/.

[12] ARM, "ARM security technology, building a secure system using TrustZone technology," http://www.arm.com, 2009.

[13] T. Alves and D. Felton, "TrustZone: Integrated hardware and software security," *ARM white paper*, vol. 3, no. 4, pp. 18–24, 2004.

[14] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using ARM TrustZone to build a trusted language runtime for mobile applications," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014, pp. 67–80.

[15] H. Liu, S. Saroiu, A. Wolman, and H. Raj, "Software abstractions for trusted sensors," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, 2012, pp. 365–378.

[16] S. D. Yalew, G. Q. Maguire Jr., and M. Correia, "Light-SPD: A platform to prototype secure mobile applications," in *Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing*, 2016, pp. 11–20.

[17] S. D. Yalew, G. Q. Maguire Jr., S. Haridi, and M. Correia, "T2Droid: A TrustZone-based dynamic analyser for Android applications," in *Proceedings of the 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, Aug. 2017, pp. 25–36.

[18] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala, "On-board credentials with open provisioning," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, 2009, pp. 104–115.

[19] Y. Gilad, A. Herzberg, and A. Trachtenberg, "Securing smartphones: a micro-TCB approach," in *arXiv preprint arXiv:1401.7444*, 2014.

[20] S. D. Yalew, P. Mendonça, G. Q. Maguire Jr., S. Haridi, and M. Correia, "TruApp: A TrustZone-based authenticity detection service for mobile apps," in *Proceedings of the 13th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, Oct. 2017.

[21] C. Manning, "Yaffs: Yet another flash file system," https://yaffs.net/archives/yaffs-nand-specific-flash-file-system-introductory-article.

[22] T. Tso, "Android will be using ext4 starting with gingerbread," http://elinux.org/Android_File_Systems.

[23] D. Salomon, *Data compression: the complete reference*. Springer, 2004.

[24] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems*, 2002, pp. 617–624.

[25] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.

[26] National Computer Security Center, "Trusted computer systems evaluation criteria," Aug. 1983.

[27] rsync, https://rsync.samba.org/, 2017.

[28] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security & Privacy*, vol. 5, no. 2, 2007.

[29] Genode Labs, "ARM TrustZone, an exploration of ARM TrustZone technology," http://genode.org/news/an-exploration-of-arm- trustzone-technology, 2014.

[30] Witekio, "NXP i.MX 53 reference BSP," http://witekio.com/cpu/i-mx-53/.

[31] Bup, https://github.com/bup/bup.

[32] Freescale, "i.MX53 multimedia applications processor reference manual," Document Number: iMX53RM Rev. 2.1, 06/2012.

[33] genbackupdata, https://linux.die.net/man/1/genbackupdata.

[34] D. Guides, "Auto backup for apps," https://developer.android.com/guide/topics/data/autobackup.html.

[35] K. Kwang, "Android flaw can disable, corrupt AV tools," Sep. 2011, ZDNet, http://www.zdnet.com/article/android-flaw-can-disable-corrupt-av-tools/.

[36] Cloud Security Alliance, "The notorious nine: Cloud computing top threats in 2013," Feb. 2013.

[37] F. Rocha and M. Correia, "Lucy in the sky without diamonds: Stealing confidential data in the cloud," in *Proceedings of the 1st International Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments*, 2011.

[38] D. G. Severance and G. M. Lohman, "Differential files: their application to the maintenance of large databases," *ACM Transactions on Database Systems*, vol. 1, no. 3, pp. 256–267, 1976.

[39] R. J. Green, A. C. Baird, and J. C. Davies, "Designing a fast, on-line backup system for a log-structured file system," *Digital Technical Journal*, vol. 8, pp. 32–45, 1996.