Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Secloud: A cloud-based comprehensive and lightweight security solution for smartphones

Saman Zonouz [b,*], Amir Houmansadr [a], Robin Berthier [a], Nikita Borisov [a], William Sanders [a]

[a] University of Illinois, Department of Electrical and Computer Engineering, 1251 Memorial Drive, McArthur Bl., Coral Gables, FL 33146, United States
[b] University of Miami, United States

ARTICLE INFO

Article history:
Received 16 June 2012
Received in revised form
31 January 2013
Accepted 4 February 2013

Keywords:
Smartphone security
Intrusion detection
Cloud computing
Energy-aware security
Real-time intrusion response

ABSTRACT

As smartphones are becoming more complex and powerful to provide better functionalities, concerns are increasing regarding security threats against their users. Since smartphones use a software architecture similar to PCs, they are vulnerable to the same classes of security risks. Unfortunately, smartphones are constrained by their limited resources that prevent the integration of advanced security monitoring solutions that work with traditional PCs. We propose Secloud, a cloud-based security solution for smartphone devices. Secloud emulates a registered smartphone device inside a designated cloud and keeps it synchronized by continuously passing the device inputs and network connections to the cloud. This allows Secloud to perform a resource-intensive security analysis on the emulated replica that would otherwise be infeasible to run on the device itself. We demonstrate the practical feasibility of Secloud through a prototype for Android devices and illustrate its resource effectiveness by comparing it with on-device solutions.

1. Introduction

Smartphone devices are rapidly increasing their popularity by offering advanced computing and connectivity functionalities. In contrary to traditional mobile phones, smartphones allow users to run a diverse set of third-party software applications. As an evidence to the massive popularity of smartphones, a recent study by ComScore Inc. indicates that over 110 million people in the United States owned smartphones in 2012 (C. reports June 2012). This represents a 47% share of the total number of mobile communication devices sold and continues to grow (C. reports June 2012).

The increasing popularity of smartphones has attracted cyber criminals to this area as well. This has given birth to various types of smartphone malware such as viruses and trojans. One can broadly classify smartphone malware into two main groups. The first group are *smartphone-specific* malware that are designed to leverage the unique software and hardware capabilities of smartphones. For instance, Cabir (Library) is a smartphone-specific worm that spreads through the Bluetooth interface of smartphones. As another example, a recent study (Schlegel et al., 2011) builds a proof-of-concept, smartphone-specific trojan that uses voice-recognition algorithms to steal sensitive information being spoken by a user. The second category of smartphone malware are *generic* malware that are designed to work on a wide range of computing devices including computers, smartphones, and tablets. Phishing attacks (Dhamija et al., 2006) are examples of

* *Corresponding author.*
E-mail addresses: s.zonouz@miami.edu (S. Zonouz), ahouman2@illinois.edu (A. Houmansadr), rgb@illinois.edu (R. Berthier), nikita@illinois.edu (N. Borisov), whs@illinois.edu (W. Sanders).

0167-4048/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cose.2013.02.002

generic intrusions that intend to steal a user's logging credentials to a website (e.g., an online banking account) by mimicking the appearance of that website. In addition to invading the privacy and security of smartphone users, smartphone malware can also form botnets that are capable of performing large-scale, coordinated attacks on communication infrastructures (Apvrille, 2010).

A key challenge in building effective smartphone security solutions is the resource limitation of smartphones. Existing security solutions for smartphones consume much resources for their operation, such as memory, storage, CPU, and battery; this compromises their usability and encourages their users to avoid such solutions. Indeed, to be effective, a security solution needs to keep a comprehensive database of malware signatures, requiring a large storage from the phone device. Also, any suspicious behavior needs to get correlated against a large list of stored signatures, consuming high processing and memory resources from the resource-limited smartphone devices. Additionally to protect users against zero-day threats, many suggest to combine different security solutions, which is not feasible for on-device deployment considering the smartphones' limited resources (Oberheide et al., 2008a). As an example of high resource utilization in smartphone security solutions, SMobile VirusGuard,[1] a popular antivirus solution for Android phones, takes 40 min and consumes 10% of the battery charge to scan a 200 MB folder on a typical Android phone (Zhao et al., 2010). To conserve resources, some solutions run a lightweight process on the smartphone device (Miettinen and Halonen, 2006; Boukerche and Notare, 2002; Yap and Ewe, 2005); this, however, affects the effectiveness and accuracy of such solutions significantly in protecting the devices against security threats (Biever, 2005). Alternatively, some solutions move the resource-intensive security analysis from the device to a network entity (Guo et al., 2004; Cheng et al., 2007) or inside a cloud (Oberheide et al., 2008b; Portokalidis et al., 2010). Due to their limited interaction with the device, such solutions have limited capabilities in performing security analysis. For instance, the cloud-based approach of Oberheide et al. (2008b) extensively analyzes a device's files inside a cloud to identify possible corruptions, yet, it is not able to identify in-memory exploitations.

In this paper, we build on our previous preliminary work (Houmansadr et al., 2011) and propose and implement Secloud, a comprehensive security solution for smartphones. Secloud emulates a smartphone device inside the cloud and runs various powerful (hence, resource-intensive) security analysis on the emulated replica, in parallel, in order to protect and assess the security of the actual device. Moving the resource-intensive analyses to the cloud enables Secloud to provide powerful protection against intrusions without exhausting the smartphone's limited resources. More specifically, Secloud emulates a registered smartphone in a virtual machine running in the cloud and mirrors all of the device's inputs and communications to the emulated environment to keep the emulated version synchronized with the actual device. This provides *real-time* and *powerful* security analysis capabilities while consuming low resources on the device itself. The real-time emulation on powerful servers (inside the cloud) allows Secloud to instrument the emulated environment with a rich set of off-the-shelf intrusion forensics and detection solutions that do not have to be lightweight, and perform run-time in-depth detection analyses. Once a security compromise is detected within the emulated environment, Secloud instructs its lightweight agent running on the device to take the required actions, e.g., to remove an infected file or to close an attacker's network connection.[2]

We have implemented a working prototype of the Secloud framework for Android smartphone devices. We instrumented Secloud on an actual smartphone device and evaluated its security performance against several real attacks such as the `Rageagainsthecage` (Rageagainsthecage, 2010) exploit, a powerful malware that is known to have infected over 50,000 devices. We also measured the resource utilization of Secloud's agent on the device and compared the results with other security solutions with similar objectives. Our results show that Secloud uses little resources on the device, while performing comprehensive and complicated security analyses by running heavyweight analyses on the emulated device inside the cloud.

1.1. *Paper's scope*

In this paper we are only interested in security threats against smartphone devices, but not those that target the underlying infrastructures such as cellular networks. We assume that typical security measures are deployed to secure the cloud environment and the Internet communications between Secloud entities. Additionally, while our cloud-based security solution can be extended to detect *any* kind of smartphone malware, the Secloud prototype presented in this paper only implements PC-based malware detectors such as anti-viruses and intrusion detectors. For instance, a more advanced version of Secloud would be able to detect malware that make physical damages to the device by emulating and monitoring the hardware behavior in the cloud. We leave such extensions as our future work plan.

1.2. *Contributions*

In summary, we make the following main contributions: 1) we design Secloud, a generic framework for smartphone security that can be used to perform various powerful intrusion analysis solutions, while imposing little resource utilization on the device. 2) We prototype a working implementation of Secloud for Android smartphone devices and measure and evaluate its performance and accuracy on a real-world testbed.

The rest of this paper is organized as follows. In Section 2, we give an overview of the Secloud architecture. Sections 3–5 describe the three main components of Secloud, mentioning different security solutions deployed by Secloud in our implementations. In Section 6, we suggest and compare two

[1] http://www.smobilesystems.com/wp-content/themes/new_smobile/datasheets/SMobile_VirusGuard_2010.pdf.

[2] As discussed later, to guarantee that the user gets aware of the intrusion, Secloud makes use of other secure channels to notify the user such as sending an email to an account that is not logged in on the device.

deployment scenarios for Secloud, and in Section 7, we discuss several issues and challenges regarding the Secloud's real-world deployment, along with their corresponding potential solutions. We present the evaluation results of the Secloud prototype implementation in Section 8. Section 9 mentions some related work, and we conclude the paper in Section 10.

2. Secloud's architecture overview

Fig. 1 shows the high-level architecture of Secloud's framework. Secloud is composed of three main components: a *client agent* running on smartphone devices, an *emulator* that runs replicas of registered smartphones in the cloud, and a *proxy server* that mirrors network traffic between the registered smartphones and the emulated replicas.

The client agent is a software that is installed on the smartphone device once the device gets registered to Secloud. The agent is composed of a lightweight mobile application, and a kernel module that provide permissions and access to low level sensor inputs. The client agent performs three main tasks: 1) it collects user and sensor inputs from the device's interfaces and sends them to the Secloud's emulator; 2) it modifies the network settings of the device to use Secloud's proxy for its network communications; and, 3) it listens for notifications from the emulator and performs the requested actions.

Each emulator resides in a cloud environment and runs an emulated replica of each device registered to Secloud. Emulated replicas contain the exact file systems and operating state of their corresponding devices and are continuously kept synchronized with them. The emulation environment runs several off-the-shelf security solutions in parallel in order to analyze the security state of the replicas. It is important to note that such a comprehensive security solution can not be deployed directly on the device due to the limitations of processing, memory, storage and battery resources. Furthermore, periodic snapshots are taken in the cloud to provide recovery capability if needed. Once a misbehavior is detected, the emulator sends a notification to the corresponding client agent to take the required actions.

Finally, the third component of Secloud is a proxy server that mirrors Internet communications of registered devices and sends them to the emulated replicas running inside the cloud. We note that the proxy's operation does not disrupt the usual Internet activity of the client. Outgoing traffic from the replica can either be discarded of buffered to enable comparison with outgoing traffic from the device.

As mentioned above, all of the registered smartphones' interactions with the outside world (e.g., user inputs, sensor inputs, and Internet communications) are sent to the emulated replica of that device. Starting from a synchronized state, the architecture maintains synchronization of the device and its replica that is required to perform in-the-cloud accurate security analyses in real-time. In Section 7, we discuss possible inconsistencies between a Secloud device and its replica and describe how Secloud minimizes their effects on the security incident analysis results.

3. Client agent

The Secloud agent is a lightweight software that runs on the smartphones registered to use Secloud service. The agent captures and logs all the information that are required to fully duplicate the registered device within the emulation environment in real-time. A straightforward solution to achieve this objective is to take an approach similar to the replay mechanism in ReVirt (Dunlap et al., 2002), in which the agent logs all inputs to the device, such as network incoming traffic and physical sensor and keyboard inputs. Replaying "all" the

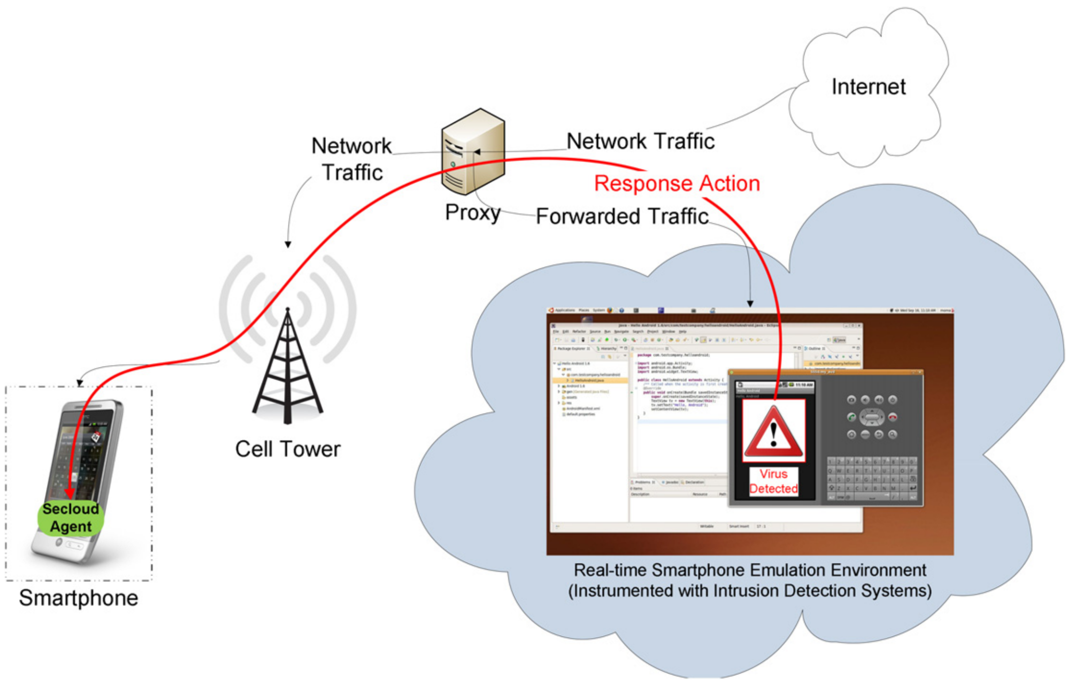


Fig. 1 – Secloud's high-level architecture.

inputs enables an accurate replay with no need to periodically run consistency checks between the device and the emulator. However, this approach is resource-intensive and results in a high volume of overhead traffic being generated by the device.

To lower the bandwidth consumption, we take an alternative approach where the Secloud agent logs only device inputs and sends them to the emulator to be replayed. Since most Internet services, e.g., webpages, have deterministic effects on the state of the device, the device and the emulator receive the same responses from remote servers in response to similar requests. For instance, if a smartphone user unknowingly clicks on a link to download a malware, the clicking event on the device would be logged, sent to the emulator and replayed in the emulator, resulting in the same malware download in the emulation environment.

To capture the physical inputs, the Secloud agent intercepts them at the kernel level, right after they have been received from the hardware by drivers. This implementation choice is based on the following reasoning. First, input interception from layers above the kernel, e.g., from the application layer, requires application-specific knowledge and should be implemented for each application separately. Second, a single hardware event might affect several applications; therefore, kernel-level input interception prevents such redundant application-level event loggings. Finally, the input interception module needs to be in a trusted domain as the reports should be trusted by the emulation environment. We note that the kernel-level interception module is the last possible target for an attack after the other security layers have been compromised.

3.1. Implementation

We implement Secloud agent software for Android smartphone devices. In particular, the agent modifies the input subsystem of Android OS to capture and log physical inputs. The input subsystem's main responsibility is to manage various input devices, such as the keyboard and pointer, that one uses to interact with the phone. These input devices are usually accessed through special hardware interfaces. Using the input subsystem, the kernel exposes the user input to the user space. The input subsystem has three main components, as depicted in Fig. 2: 1) the hardware drivers; 2) the `input_core` subsystem that interacts with the low-level hardware drivers, and manages the interconnections among drivers and handlers; and 3) the `evdev` event handlers that provides an interface to the user space.

Secloud's intercepting agent on the device captures the physical inputs once they have been processed by the drivers and are ready to be passed to the handlers (see Fig. 2). The agent captures each event as an array of parameter values. The first field is a timestamp storing the time when the event occurred. As we will discuss later, the absolute timestamp values are not crucial, because Secloud uses event time only to maintain the inter-event interval consistency while replaying the events. The second field is the type, which shows the generic type of the event being reported, such as a key press, button press, or relative motion. The last two fields store the event code and the event value. The code field determines which of the various buttons or axes are being manipulated, while the value field stores what the state or motion is. For

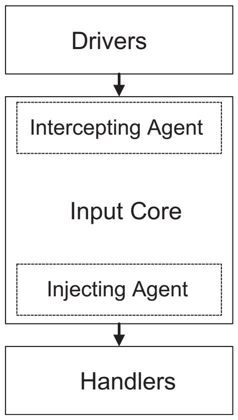


Fig. 2 – The modified linux input subsystem.

example, if the type is a key or button, the code saves which key or button it is, and the logged value declares whether the button has been pressed or released.

If continuous network availability (e.g., a data plan) is provided, once the Android OS on the device has booted up, Secloud's intercepting agent initiates a kernel-level socket that connects to the emulation environment. The socket is later used to send all the logged events in real-time to the emulator, while the device is actually used by the client. Within the emulator, Secloud's injecting agent receives the events and passes them to the event handlers preserving the logged events' orders and timings. The real-time capture and forward approach makes it harder for attackers to disable or corrupt the agent's reporting capability before the logs of malicious events have been sent and replayed in the emulator. If there is no Internet connectivity on the smartphone device, the logs are stored on the phone's SD card and are sent later to the emulation environment once the device is connected to the network (e.g., through WiFi or USB connection to a PC). In this case, to ensure that logs' integrity is not compromised, Secloud makes use of hash function chains before storing the logged events. In particular, to store the sequence of events $e_1, e_2, \ldots$, Secloud stores the following (*event, hashed_value*) pair sequence: $(e_1, h(e_1||0)), (e_2, h(e_2||e_1)), \cdots$ where $h(i||j)$ is a cryptographic one-way hash function on the concatenated string composed of i and j. We note that while the hashing process prevents an attacker from tampering with the logs, an attacker could still erase the logs to hide malicious activities. Note that the capture-and-forward approach in real-time partially addresses this issue, because the transition from secure to insecure state would still leave traces in the logs before the malicious deletion of events occurs. Another barrier that we plan to implement as future work is a local heartbeat between the kernel agent and the log storage to periodically check that no log has been deleted.

To save network or storage resources on the device, the agent compresses data before they are sent or stored. In particular, we use the Run-Length Encoding (RLE), which is a lossless data compression algorithm. RLE is very suitable for compression of event reports on the device, since in practice, several subsequent events' parameters on the device usually are of the same value, i.e., the type field. Furthermore, being a very efficient algorithm, RLE does not cause much performance overhead.

4. Secloud's emulator

The emulation environment of Secloud hosts the smartphone replicas. The emulation environment also hosts different security solutions such as virus scanners and intrusion detectors that run in paralel over the replicas corresponding to different actual devices. An emulated replica is kept *synchronized* with its corresponding smartphone device; this is done through receiving the user inputs from the corresponding client agent. We say that a device and its replica are synchronized if they both have the same OS configuration, the same file system (hence, the same installed applications), and the same set of running applications. Since the user inputs to the actual device are also replicated in the emulated replica, the replica is expected to remain synched with the device if it boots from a synchronized state. However, the device and its replica may go out of synchronization occasionally for different reasons such as a security compromise or software failures. To detect dysynchronizations, the emulator frequently checks the hash of the file system and tries to re-synchronize the replica by communicating with the device's user agent. The emulator can also notify the owner, as described in Section 4.2, if the resynchronization is failed due to a security compromise.

We have implemented the reception and replay of user inputs using the same kernel agent described in the previous section. A set-bit in the agent configuration file tells the agent to whether run in the *collection mode* or the *injection mode*. In the collection mode, Secloud event logging engine, intercepts and logs all inputs to the device. In the injection mode, intercepted events are collected by a network socket and written to the Android input subsystem as if it were a new driver. The timestamps on each event are used to delay the injection and maintain a consistent inter-event interval. This allows Secloud to reproduce an accurate user behavior on the replica and to reduce the chances of the replica going into a state inconsistent with the cell phone device.

Unlike an actual device, the emulation environment is not resource-limited, and hence can be used to deploy multiple security solutions concurrently to monitor smartphone replicas for various types of compromises. Furthermore, Secloud's architecture enables out-of-the-box security solutions, e.g., virtual machine introspection, which could not be tampered with by the attackers once the device or replica has been compromised. Indeed, to improve the overall detection accuracy, Secloud makes use of several off-the-shelf IDSes, which we review in the following sections.

4.1. *Security analysis deployed by the emulator*

To achieve the highest protection, Secloud employs different host-based and network-based security solutions on the emulated smartphones inside the cloud, as described in the following.

4.1.1. *Virus scanning*

Secloud uses antivirus solutions to recursively monitor all files on the device in order to find known malware, such as viruses or Trojans, using a traditional pattern-matching technique. Unlike cell phone-specific antivirus solutions, which consider resource limitation of such devices, cloud-based emulation allows Secloud to run powerful antivirus tools with high-frequency file system check ups. In our prototype, Secloud employs the *ClamAV*[3] tool, which uses an extensive database of malware signatures (currently 846 K signatures) as well as signatures for safe Internet browsing (currently 763 K signatures). Secloud periodically mounts the smartphone's different images, such as/system and/sdcard, using Ubuntu's built-in `mount` command and the open-source `unyaffs2`[4] utility. An advantage of performing out-of-the-box file system checking is to prevent attackers from being able to tamper with the antivirus.

4.1.2. *File integrity checking*

When a system is compromised, the attacker may modify or read sensitive parts of the file system to keep the doors open for future access to the device, or to evade detection. A file integrity checker can check for modifications of important files by generating a digest of those files, e.g., using cryptographic hash functions, and by periodically checking the files to validate their integrity. Due to high resource consumption, it is impractical to run a file integrity checker on smartphone devices. Secloud does not have that limitation and is configured to run the *Samhain* file integrity checker (Wotring et al., 2005) on the mounted file system images every 20 s.

4.1.3. *System-call monitoring*

Virus downloads and file modifications usually happen before or after the device is compromised. More specifically, a user might get tricked into downloading a malware, resulting in a compromise, or the attacker may download malware after compromising the device. To detect vulnerability exploitations while they occur, we have modified the Android kernel on the replica to record syscalls while the emulator is operating. The logs are sent out of the emulator in real-time so that a syscall-based detection solution (Forrest et al., 2008) can monitor for signatures, i.e., illegitimate syscall traces that represent malicious activities. For instance, the well-known `rageagainstthecage` Android malware (Rageagainsthecage, 2010) forks more than a thousand processes (calling the fork syscalls) within a few seconds to bypass permission protection and to gain root access on the device.

4.1.4. *Network intrusion detection and response*

Cloud-based emulation also enables Secloud to do in-depth packet inspection on incoming and outgoing network traffic. This helps to monitor the device for malformed payloads and anomalous behaviors such as network scanning and botnet behavior. In our implementation, Secloud uses the Snort (Roesch, 1999) network-based IDS, which stores its signature database in the cloud environment and inspects egress and ingress traffic of the emulated replicas.

If Secloud is deployed as a subscription-based service, it is very likely to monitor several instances from each smartphone model. This enables cooperative intrusion detection operations to identify widely spread threats. For example, smartphone viruses propagating from one device to the other would allow Secloud to identify the same anomalous behavior

[3] http://www.clamav.net/lang/en/.
[4] http://unyaffs.googlecode.com/files/unyaffs.

on many devices in a short period of time. As a future work, we plan to investigate how to correlate IDS alerts from different device samples in the cloud to reduce the overall false positive and false negative rates.

4.2. *User notification*

Once a security threat is detected by the emulator's security solutions, the corresponding device should be notified to take the required actions. We suggest the following mechanisms:

4.2.1. *Informing Secloud's client agent*

The immediate way for notifying the user is to send a message to the Secloud client agent running on the corresponding device. The Secloud client agent can notify the user through a pop-up message about the detected security threat. Also, the emulator can command the Secloud client agent to perform certain actions to mitigate the detected compromise, e.g., to delete some files or replace them with a previous version.

4.2.2. *Email notification*

To backup the previous mechanism, i.e., in case the client agent is compromised or disabled, the Secloud emulator can use other channels to respond to a detected threat. In particular, the emulator can send an email through a secure external service, e.g., Gmail, to the client. As an alternative mechanism, each user can setup an emergency contact list with the Secloud system, which get contacted once a compromise is detected. As another alternative, Secloud can directly notify the mobile service provider of the infected device. The service provider can analyze the Secloud reports from different smartphones in order to detect fast-spreading viruses or worms.

4.3. *Response and prevention*

Once the client agent is notified about an identified compromise, an appropriate action needs to be taken to mitigate the compromise and to restore the device to its secure state. The following summarizes the response and recovery actions taken by a Secloud client agent.

4.3.1. *File removal*

The emulator can instruct the corresponding client agent to remove detected infected files, or malicious applications.

4.3.2. *Process termination*

In case the malicious executable is successfully launched as an application/process on the device, the emulator can instruct the corresponding client agent to kill the misbehaving process.

4.3.3. *Periodic backups*

The Secloud emulator can periodically backup the emulated replica, which can be used to restore the device in case of malicious corruptions. In fact, performing this backup does not impose any additional network communication to the device, which is the case in traditional smartphone backup services.

Table 1 – A sample list of response and prevention actions performed by a client agent.

Incident	Response actions
Malformed incoming packet detection	Restore clean `system.img` and`user-data.img` images, and reboot the smartphone device
Network-based device misbehavior detection	Terminate the connection and quarantine the application which receives the packet
Virus/Malware detection on/data directory	Remove the file and block the communication with the downloaded site
Virus/Malware detection on/system directory	Restore a past clean `system.img` snapshot and block the downloaded site
Misbehaving application-level process	Kill the process or terminate the corresponding service and uninstall the application
Misbehaving root-level process	Kill the process, restore the `system.img` image and reboot the smartphone device

4.3.4. *Network filtering*

The file removal and process termination mechanisms mentioned above help in stopping an occurred compromise. In order to prevent known compromises, the emulator can instruct the client agent to terminate some network connections that have malicious patterns.

4.3.5. *Device quarantining*

By receiving frequent Secloud reports for a device within a short period of time, the service provider can temporarily quarantine the device by blocking its network communication to prevent spreading of the compromise to other devices, until the compromised device is restored to a secure state.

In our current implementation, all the mentioned response and prevention mechanisms have been implemented, except for the device quarantining, which needs to be deployed by a service provider. In particular, the periodic backups inside the cloud are implemented by taking periodic snapshots of the virtual machines running the emulated replicas. In our implementation, we used VirtualBox[5] as a virtualization solution. Table 1 shows a list of the actions taken by the client agent, as instructed by the Secloud emulator. In particular, we can see that different actions are taken for malicious files detected inside/`data` and/`system` directories. This is because during the normal usage, the/data directory gets updated with high frequencies, i.e., higher than the snapshot frequencies, so restoring to a previous snapshot may lead to data inconsistencies.

5. Secloud's proxy

The third component of Secloud is a proxy server that duplicates a device's incoming traffic to their corresponding emulated replica inside the Secloud cloud. It also blocks the replica's outgoing traffic to avoid request duplication from the point of view of third parties (e.g., to avoid double-purchasing an item from an online shopping Website).

[5] https://www.virtualbox.org/.

As mentioned before, the client agent installed on the smartphone configures the device to use the dedicated proxy for its Internet traffic. Rather than a traditional HTTP proxy, we opted to use a SOCKS proxy (Leech et al., Apr. 1996) to enable the client agent to proxy any type of IP traffic.

We note that an alternative solution to deploy the proxy in the cloud is to leverage the cellular service providers. In other words, a service provider can forward all of the traffic of a registered user through its cellular network to the emulation environment of Secloud. This approach eliminates the need for proxy settings in the registered devices and reduce the latency added due to proxying the Internet traffic. However, deploying this by a service provider may be a more expensive solution. Moreover, this approach has to fall back to the first option of a client-connected proxy when the device switches from using the carrier network to a traditional WiFi connection.

Finally, it is important to mention that a smartphone can have communication channels other than the carrier's cellular network and WiFi connections, e.g., infrared and the Bluetooth interfaces. Since these connections are local and out of reach for the proxy, the client agent is designed to capture them and send them directly to the Secloud emulator. Note that these type of connections occur much less frequently than the cellular and WiFi communications, hence they do not impose large traffic loads to the device.

6. Deployment scenarios

We suggest two different deployment scenarios for Secloud.

A) *Personal deployment*. An individual can deploy a personal Secloud system to protect her/his smartphones. To do this, one needs to run the Secloud emulator on a computer machine, which could be a personal computer or a leased cloud server. She also has to deploy the Secloud proxy on the same machine that runs the emulator. For any smartphone device that needs to be protected by this personal Secloud system, the user should install the Secloud client agent on the device and to configure it to use the personal Secloud emulator and proxy.

B) *Commercial deployment*. In this scenario, the Secloud system is deployed and maintained by a third-party, e.g., a for-profit company. The company runs Secloud emulators on a server farm. The company also runs proxy server(s) that duplicate the traffic of the users to the emulator. Smartphone users can subscribe to this service, e.g., by paying a monthly fee, and use the service by installing the client agent provided by the company. Such commercial deployment can be either performed by cellular service providers or by companies that target users across different service providers.

6.1. *Comparison*

The personal approach provides the maximum privacy protection to the users but is likely more expensive compared to a large-scale commercial deployment. Also, a personal deployment should be performed by someone with the required technical expertise to install and maintain the system. On the other hand, the commercial deployment can cut the cost of maintaining the system and purchasing security solutions for the users by distributing expenses and efforts among many users. Also, the third-party deployment would be more suitable for users with minimum technical knowledge.

7. Discussions

In this section, we discuss current limitations and potential solutions regarding practical deployment of the Secloud framework.

7.1. *File-system consistency*

Even though, in the Secloud framework, all the inputs to the smartphone device are replayed in the emulation environment, the file system on the device and the replica could possibly go out of synchronization for various reasons, such as accidental transient hardware or OS errors. To solve that problem, the Secloud's agent periodically (e.g., once per day, when the phone is getting recharged) computes hashed values of folders on the device and sends them to the emulator for comparison. In the case of a mismatch, Secloud notifies the client that a resynchronization is needed.

To minimize the amount of data being transferred from the device to the emulator during a resynchronization process, Secloud follows a recursive hierarchical integrity check so that only mismatched files are uploaded. The algorithm works as follows: starting from the root directory/on the device, the Secloud's agent computes and sends the hashed value of the whole folder for comparison. If a mismatch is detected, the agent proceeds to check subdirectories recursively until the source of the inconsistency is found. Then the current copy of the file(s) or folder(s) is sent from the device and replaces its existing version, if there is any, on the emulator. Once the device and the emulator have been synchronized, Secloud reboots both entities to make sure their memory images are also at the same state.

7.2. *User privacy*

As discussed before, user input and data stored on the device are sent to the emulator to keep the replica in the cloud well-synchronized. For a personal deployment of Secloud, this has limited privacy concerns since the emulator is owned and moderated by the smartphone owner. However, in the case of a commercial deployment, users may be concerned about the privacy of their personal information being sent to a third-party controlled emulator. In fact, this is a common problem with many Internet services, like the Amazon EC2[6] computing cloud service or online data backup service including Google Android backup,[7] Apple iCloud,[8] or Dropbox.[9] The core idea behind the operation of those services is a tradeoff between

[6] aws.amazon.com/ec2/.
[7] http://code.google.com/android/backup/index.html.
[8] https://www.icloud.com.
[9] www.dropbox.com/.

usability and security. If the benefits offered by those services meet a certain level of trust, then users will adopt the solution because the service offers strong security protection and the likelihood of a privacy breach is acceptably low, given the adequate privacy policy agreement and security insurance offered by service providers. We believe that Secloud fall in the same category. As an example, a company maintaining a large volume of cell phones for their employees would find the security features of Secloud attractive in order to outsource intrusion detection and keep devices protected at a reduced cost, even if this means sending data to the cloud. Of course, this does not mean that less scrutinity should be applied on the security of the architecture.

Additionally, users can be given the option to choose specific applications to be monitored or not by Secloud. For example, a client may choose to have only newly installed applications monitored, but leave a trusted set of applications that deal with sensitive information (e.g., financial account management applications) out of the replication process.

7.3. *Environment resiliency*

Leveraging an emulation environment to detect security issues involves the risk of missing malware crafted to change their behavior under emulated environments. Few years ago, a study showed that 4% of malware targeting PCs included instructions to detect emulated environments and to change their behavior accordingly (Chen et al., 2008). In the context of our study, a malware that intends to infect the cell phone device and to remain invisible to the replica running in the cloud would likely cause the device and the replica to become unsynchronized. To keep the illusion of the replica and the device being synchronized, the malware would have to succeed in infecting the client agent at the kernel level. While possible, the Secloud architecture makes this type of attack significantly more complex and expensive for adversaries.

7.4. *Encryption*

Some smartphone applications use encryption in their Internet communications, e.g., a web browser connecting to an HTTPS destination. In order for the Secloud emulator to be able to analyze an encrypted connection, a Secloud replica should obtain the appropriate encryption credentials used by its corresponding Secloud device. For this, we have the Secloud agent to send all the credentials for encrypted traffic to the emulated replica, to be able to decrypt and analyze the corresponding encrypted traffic. An alternative solution would be to deploy this at the application-level on the device, by having an application that uses encrypted connections, e.g., a web browser, to send a decrypted version of its encrypted traffic to the emulator. Our measurements show that a small fraction of a typical smartphone's Internet connections are encrypted, so this increases the utilized bandwidth by the device only slightly.

7.5. *Attack evasions*

One way to defeat Secloud is to replace the device drivers and then fool the on-device agent with malware that inaccurately reports hash values of the file system. The mobile device could then host spyware while spoofing the cloud into thinking that no changes have been made. In order to get root-level privilege and get access to device drivers for replacing them, the attacker would need to go through several adversarial steps, such as vulnerability exploitations and heavy system modifications. We assume that starting from a clean state, the cloud-based solution, which instruments the emulator with many monitoring solutions, will identify at least one of those steps, and consequently, Secloud may decide not to trust the on-device agent anymore depending on the detected security incident. This is why Secloud also uses a backup email notification system to warn users of an intrusion even if the device no longer respond to instructions given by the monitoring solutions in the cloud.

8. Evaluation

The goal of this section is to validate the Secloud approach under realistic conditions. More specifically, we study the following three questions: 1) can known and unknown threats be detected by this approach with better accuracy than that of traditional cell phone security solutions? 2) How much computational and memory resources are saved by this approach, and how much network bandwidth is required? 3) Finally, what would be the cost to deploy this approach at large scale using current commercial cloud services? We address those three questions through empirical experiments in the following subsections.

For the experiments, we emulated the Motorola Droid smartphone with a 16 GB SD card using a virtual machine in the cloud. In particular, the virtual machine used an Intel i7 CPU 3.07 GHz, and 3.0 GB of the system's memory. The operating system is a 64-bit Ubuntu with the Linux 2.6.32 kernel. We built the Android's phone emulator directly from the Android's kernel source code (version 1.5[10]). In the experiments, we found that each emulator instance took (on average) 210 MHz of CPU and 240 MB of the VM's memory.

8.1. *Accuracy*

To evaluate the detection capabilities of Secloud, we replicated the DroidDream malware that spread through 50 Android applications before being remotely blocked by Google in March 2011 (Perez). DroidDream works by hijacking legitimate applications to gain root access on cell phones and to run malicious payloads. We chose to develop the experimental malware by infecting the `Terminal Emulator` application (The Terminal-Emulator App, 2010), because it enables us to easily perform a variety of illegitimate actions through a set of shell scripts. From the two exploits embedded within the DroidDream malware, we used the `Rageagainstthecage` exploit (Rageagainstthecage, 2010). It exploits a vulnerability in the Android Debug Bridge (ADB) process which is used to

[10] Our solution will be applicable to all of the recent Android smartphone models, because Secloud enhances the Linux kernel's input subsystem that is used in recent Android operating systems.

communicate with an emulator or device instance. Before Android 2.3, this process did not check the return value of the setuid and setgid function calls when dropping privileges after being launched. The exploit forks itself enough times to overflow the maximum number of ADB children. As a result, any new ADB process fails to drop privileges and remains root.

We also implemented additional exploits to represent various attack consequences that are usually performed by cell phone malware. The different attack scenarios are described in the following list.

– `Root Escalation`: The root privilege domain on the device is compromised by the rogue application through the Rageagainstthecage exploit.
– `Anti-Virus (AV) Neutralization`: The antivirus application running on the cell phone is killed and removed from the system.
– `K-Malware Download`: The attacker downloads a known malicious application to/system on the device.
– `U-Malware Download`: The attacker downloads an unknown malicious application to/system on the device.
– `Data Exfiltration`: The attacker scans the file system and parses the contact list to extract personal information.
– `Premium SMS`: The application secretly sends SMSes to a premium rate number belonging to the attacker (Kirk).
– `Contact Removal`: All contacts from the contact list are removed by the attacker.
– `Agent Neutralization`: The attacker modifies the routing table to disrupt communication between the Secloud agent and the emulation environment.

Secloud enables monitoring of smartphones using a comprehensive set of IDSes running in parallel. In our experiments, none of the IDSes, except Zoner, were lightweight enough to be run on the device, and hence had to be run in the cloud. We measured how much the overall intrusion detection accuracy is affected as a result of Secloud's deployment compared to a traditional lightweight mobile antivirus application. Table 2 shows a detailed comparison of the detection capabilities of the traditional approaches and Secloud. The rows are the attack scenarios detailed above, and each column represents a specific sensor. In particular, the first column shows the results for the popular Zoner Antivirus v1.0.5 for Android devices, which successfully detected a downloaded known exploit when it was scanning the SD card directory on the device. In the cloud, the ClamAV antivirus detected the same exploit by searching through the emulator's mounted SD card image. By monitoring the network traffic, Snort identified the data transfer from the smartphone to the known malicious end-point during the data exfiltration attack. The root escalation was successfully detected by the syscall-based behavioral detection system as it called the fork system call 1042 times within a second that was marked by the IDS as a misbehavior. Finally, Samhain, a file integrity checker, triggered an alert when sensitive directories and files were modified or accessed and also when a process got killed or launched. As shown in the table, each attack vector in our experiments was detected by a particular IDS, and hence by Secloud. Facilitating deployment of all IDSes simultaneously increases the detection capability of Secloud up to the union of the individual IDSes. In summary, given the set of IDSes that we had deployed, Secloud successfully detected all the attack scenarios except the premium SMS attack, which was not detected by any of the deployed IDSes in the cloud. We note that this type of attack does not violate rules monitored by the current set of IDSes, but it would be simple for Secloud operators to add a new detector to cover this threat, without any need to involve users.

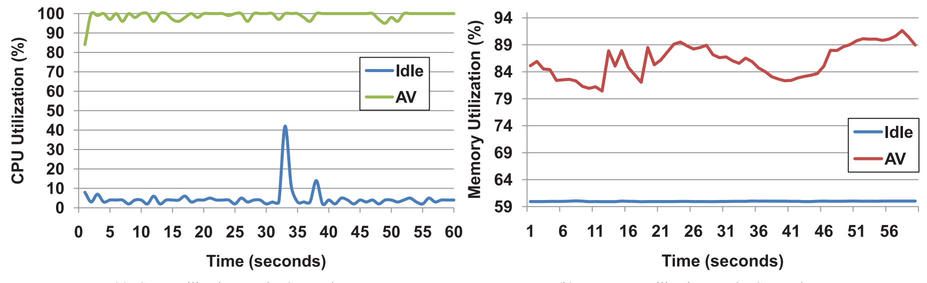
8.2. *Efficiency*

In addition to the detection accuracy improvement, we evaluated the performance of our Secloud implementation, and compared it to a conventional smartphone security solution. In particular, we measured CPU usage, memory consumption, network overhead, and battery usage. We used the adb tool to connect to the device and emulator in order to periodically read and log information stored in the `/proc/stat` and `/proc/meminfo` directories. We also used the `SystemPanel` application to log battery-related reports.

We first measured how much of the system resources a typical smartphone antivirus tool, i.e., Zoner Antivirus, needs while scanning the device for malware. We used the `dex2jar` tool to reverse-engineer the Zoner application and extract the Java source files. We found out that Zoner was matching the files against a Java hashmap (`Scanner.java`), including only 61 virus signatures. In comparison, ClamAV holds a database of 846 K signatures. For a single run, Fig. 3(a) illustrates device's CPU utilization by the on-device Zoner Antivirus (98.85%) and by Secloud (4.63%) when monitoring operations

Table 2 – Intrusion detection accuracy comparison.

	Detection solutions					
	Zoner AV	ClamAV	Snort	Behavioral	Samhain	**Secloud**
Root escalation	×	×	×	√	×	√
Binary replacement	×	×	×	×	√	√
K-malware download-1	√	√	×	×	√	√
U-malware download-2	×	×	×	×	√	√
Data exfiltration	×	×	√	×	×	√
Premium SMS	×	×	×	×	×	**×**
Removing contacts	×	×	×	×	√	√
AV neutralization	×	×	×	×	√	√



(a) CPU Utilization on the Smartphone (b) Memory Utilization on the Smartphone

Fig. 3 – Performance analysis of a traditional intrusion detection technique.

are running in the cloud (Idle). It is worth highlighting that during the Zoner scanning time, the usability of the device was significantly affected due to Zoner's high resource consumption overhead (94.22%). Fig. 3(b) shows the memory consumption of both cases. Running the antivirus solution on the device increases the memory consumption from 59.97% to 86.03%, i.e., 26.06% overhead, on average.

This experiment led us to measure how much the phone's usability was affected as a result of the virus scan. Before doing each experiment, we fully recharged the phone's battery. Fig. 4(a) shows the time required to scan folders with different sizes. As shown in the figure, for folders larger than approximately 1.5 GB, the phone became unavailable for about 1 min. A full SD card scan took 112 min to complete. For Secloud, the full SD card check with ClamAV took about 29 min; however, the scanning was done in the cloud, and the phone had no usability limitation during the scan. From a usability point of view, it is also important that the security solution does not consume much battery power of the mobile phones. Fig. 4(b) shows how much battery drain each virus scan causes. As shown in the figure, the battery drain does not grow linearly with folder size. The full SD card scan consumed about 40% of the fully charged battery and caused a 380 mV drop (from 4117 to 3737 mV).

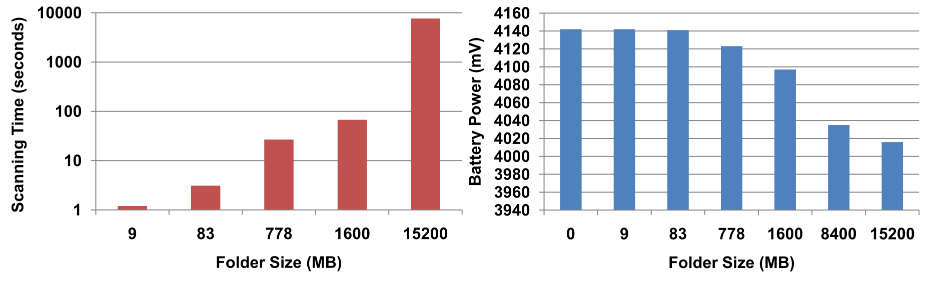
We also evaluated how much overhead Secloud causes in the emulation environment. Fig. 5(a) shows the CPU utilization for 1) VM, when the original virtual machine is up (10.96% on average); 2) Idle, when the virtual machine is running the emulator that was idle (27.08% on average); and 3) IDS, when the emulator is running and Secloud performs periodic security checks (every 15 s) using different IDS sensors (44.00% on average).

As shown in the graph, except during the security checking intervals, the CPU utilization was fairly low, enabling the concurrent running of several emulators within a single virtual machine. Additionally, Fig. 5(b) shows the memory consumption for the three cases mentioned above. Running the idle emulator itself caused a 3.46% memory increase (from 35.93% to 39.39%), while the frequent security checks increased the memory usage to 41.17%, i.e., by 1.78%.

Finally, we measured the network bandwidth consumption by Secloud as it sends all physical inputs on the device to its mirror in the cloud. Since the network overhead heavily depends on the amount of physical inputs, we measured the bandwidth consumption for the following four different usage profiles representing typical smartphone usage profiles. Fig. 6 shows the results when the user 1) does not interact with the phone; 2) browses the Web and reads the contact list; 3) types a text message, and 4) plays a game with constant use of the touch screen. As expected, the maximum network traffic is transferred when the user heavily uses the touch interface, since sliding a finger on the screen involves a large volume of discrete events to simulate a continuous motion.

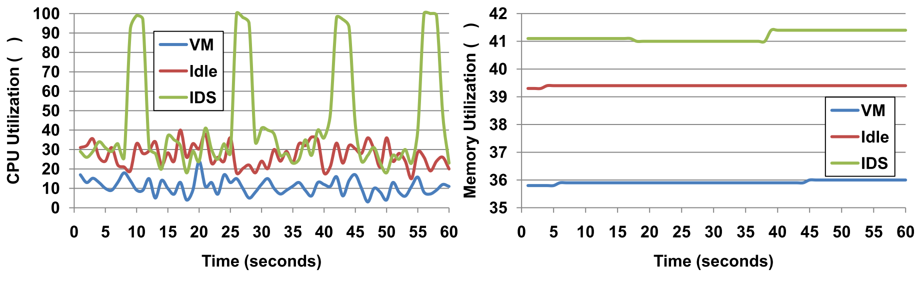
8.3. *Feasibility*

To be used in practice as a subscription-based service, Secloud needs to be cost-efficient for the end customers. We did a preliminary feasibility analysis to find out approximately how much the emulation of each smartphone device would cost, on average, if the cell phone carrier provider decided to use the Amazon EC2 Cloud as the emulation platform. Assuming that 10,000 smartphones intend to use the Secloud service, we used Amazon's monthly calculator to compute the costs for each of their computing instance types. Each row in Table 3 shows the results for a specific instance type (which is described by the second and third columns). The fourth column reports how many emulator instances could be launched on a single instance (virtual machine), and the fifth column



(a) Time Requirement for a Virus Scan (b) Battery Drain due to a Virus Scan

Fig. 4 – Performance analysis of a traditional intrusion detection technique.



(a) CPU Utilization in the cloud (b) Memory Utilization in the cloud

Fig. 5 – Performance analysis of Secloud.

shows the required number of instances to support service to 10,000 smartphones. The total costs for instance types are shown in the last column.

9. Related work

Computer intrusions, such as viruses, have been impacting end users and system operators for over two decades (Porras et al., 2010) and numerous solutions for detection and defense have been proposed (Kim and Karp, 2004; Kreibich and Crowcroft, 2004; Singh et al., 2004; Zou et al., 2005). With the increasing popularity of smartphones, smartphone-specific vulnerabilities have attracted attackers but also the research community. Initially, some studies (Dagon et al., 2004; Dunham, 2009; Racic, 2006) attempted to understand the threats and investigate the behavior of emerging attacks. Malware for Android usually exploits system vulnerabilities to get root access, and some malware succeeded in being distributed through the official Android application repos (2011). Motivations of malware authors include making money (by sending SMSes or making calls to paid services), stealing personal information, or disrupting the network (through denial of service attacks) or user activity (by locking the phone or deleting user data). Dunham (2009) analyzes a threat model for smartphones, and proposes several potential defense mechanisms. Dagon et al. (2004) overviews various possible malicious attack types against smartphones and categorizes adversarial goals. Racic (2006) explains a particular smartphone attack that happens through exploitation of an SMS system vulnerability that can later be used to exhaust available resources, e.g., the battery.

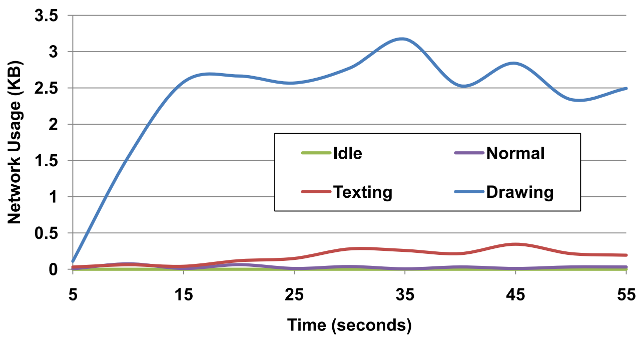


Fig. 6 – Network consumption by Secloud.

The rapidly increasing number of mobile malware strategies prompted antivirus editors to offer solutions tailored for mobile devices (Kaspersky mobile security, 2011). However, those solutions are severely limited by the low computational power of smartphones and the need to preserve battery life. Researchers introduced various alternative solutions to mitigate the malware threat without having to heavily instrument smartphones. The architecture of the SmartSiren system (Cheng et al., 2007) enables smartphones to periodically send activity logs to a centralized server that performs statistical intrusion detection operations and sends alerts to phones detected as compromised. VirusMeter (Liu et al., 2009) relies on another strategy by building power consumption profiles for smartphone applications and then running an anomaly-based intrusion detection system on those profiles, assuming that malware will generate different power consumption signatures. Kirin (Enck et al., 2009) focuses on certifying Android applications at install time to enable users to make more informed decisions regarding the security risk of smartphone applications. Kirin achieves that goal by enhancing Android security permissions with a lightweight security requirement checking mechanism.

Recently, there has been several complex host-based smartphone-specific solutions presented to address the real-time on-device intrusions detection problem (Dietz et al., 2011; Enck et al., 2010; Bugiel et al., 2012). For instance, Quire (Dietz et al., 2011) concentrates on detecting sensitive data disclosure attacks where low-privileged (and malicious) applications can trick privileged applications to access confidential personal data and send them back to the malicious application. Quire tracks all the IPC calls and blocks such call chains with potential data disclosure possibilities. L4Android (Lange et al., 2011) makes use of virtualization to separate on-device security solutions from potentially vulnerable applications as well as to support guaranteed data provenance. The main barrier in real-world deployment of such on-device virtualization-based security solution is their performance and efficiency on resource-limited smartphones, and hence their usability. AppInspector (Gilbert et al., 2011) takes a more fundamental approach to perform static binary analysis of individual applications and detect potential security and privacy violations; however, such static analysis techniques can result in high false positives and ignore runtime information in detection of adversarial intrusions.

Oberheide et al. (2008b), describe a virtualized cloud security service that offers remote file checking and remote

Table 3 – Feasibility analysis of Secloud deployment using the amazon EC2 cloud service.

Instance types	CPU (GHz)	Memory (GB)	Num of phones	Num of instances	Total cost ($)
Small	1.2	1.7	6	1750	108,885
Large	2.4	7.5	11	875	217,770
Extra-large	9.6	15	46	219	109,009
High-memory extra-large	7.8	17.1	37	269	98,454
High-memory double extra-large	15.6	34.2	74	135	98,088
High-memory quadruple extra-large	31.2	68.4	149	67	196,176
High-CPU medium	6	1.7	7	1412	175,584
High-CPU extra-large	24	7	29	343	170,233
Cluster compute quadruple extra-large	40.2	23	96	104	121,804
Cluster GPU quadruple extra-large	40.2	22	92	109	167,554

application behavior analysis. Its main advantage over traditional antivirus solutions is that it saves energy, CPU, and memory resources. The difference from our approach is that user input is not taken into account during checking of the dynamic behavior of new applications, which limits the behavioral detection coverage. The idea of capturing and replaying user input has been suggested by Flinn and Mao for security purposes but not implemented or evaluated. The work closest to our solution is (Portokalidis et al., 2010), in which the authors describe how they built a full replica of the phone in the cloud and instrumented it with security solutions. The difference is that their tracing and replay occur at the application level through execution trace. The advantage over our approach is the elimination of nondeterministic inputs, but it involves significantly more overhead to transmit compressed execution traces than to replicate only user input. We first proposed the idea of input-driven cloud-based intrusion detection in Houmansadr et al. (2011). This paper builds on this preliminary idea by providing the full design, details about the implementation, and evaluation results.

10. Conclusions

In this paper, we presented Secloud, a cloud-based service to provide security and intrusion tolerance to resource-limited smartphone devices. Secloud provides a powerful, yet resource-friendly, protection for smartphones by performing the security analysis on an emulated version of the devices, running inside a cloud. We show that our platform is able to leverage different types of security solutions to analyze smartphone devices. We propose a personal and a subscription-based deployment scenario for Secloud. As the next important future work step, we are planning to address the intrusion detection uncertainty problem so that Secloud provides best-effort protection against smartphone attacks that succeed to evade the monitoring solutions during some of their penetration steps.

REFERENCES

Android application repository, http://www.androidpolice.com/2011/03/01/; 2011.
Axelle Apvrille. Symbian worm Yxes: towards mobile botnets?, <http://www.fortiguard.com/sites/default/files/EICAR2010_Symbian-Yxes_Towards-Mobile-Botnets.pdf>.
Biever C. Phone viruses: how bad is it?, http://www.newscientist.com/article.ns?id=dn7080; 2005.
Boukerche A, Notare M. Behavior-based intrusion detection in mobile phone systems. Journal Parallel & Distributed Computing 2002;62(9):1476–90.
Bugiel S, Davi L, Dmitrienko A, Fischer T, Sadeghi A-R, Shastry B. Towards taming privilege-escalation attacks on Android. In: Proceedings of the 19th Annual Network & Distributed System Security Symposium; 2012.
C. reports June 2012 U.S. mobile subscriber market share, http://www.comscore.com/Insights/Press_Releases/2012/8/comScore_Reports_June_2012_U.S._Mobile_Subscriber_Market_Share.
Chen X, Andersen J, Mao Z, Bailey M, Nazario J. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In: DSN. IEEE; 2008. p. 177–86.
Cheng J, Wong S, Yang H, Lu S. Smartsiren: Virus detection and alert for smartphones. In: Proceedings of the 5th international conference on mobile systems, applications and services. ACM; 2007. p. 258–71.
Dagon D, Martin T, Starner T. Mobile phones as computing devices: the viruses are coming! IEEE Pervasive Computing 2004;3:11–5.
Dhamija R, Tygar JD, Hearst M. Why phishing works. In: Conference on Human factors in computing systems (CHI 2006); 2006.
Dietz M, Shekhar S, Pisetsky Y, Shu A, Wallach DS. Quire: lightweight provenance for smart phone operating systems. In: 20th USENIX Security Symposium; 2011.
Dunham K. Mobile malware attacks and defense. Elsevier Inc.; 2009.
Dunlap GW, King ST, Cinar S, Basrai MA, Chen PM. Revirt: enabling intrusion analysis through virtual-machine logging and replay. In: Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI); 2002. p. 211–24.
Enck W, Ongtang M, McDaniel P. On lightweight mobile phone application certification. In: Proceedings of the 16th ACM conference on computer and communications security. ACM; 2009. p. 235–45.
Enck W, Gilbert P, Chun B-G, Cox LP, Jung J, McDaniel P, et al. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10. Berkeley, CA, USA: USENIX Association. p. 1–6. URL, http://dl.acm.org/citation.cfm?id=1924943.1924971; 2010.
J. Flinn, Z. Mao, Can deterministic replay be an enabling tool for mobile computing?, Proceedings of the 12th workshop on Mobile Computing Systems and Applications (HotMobile).
Forrest S, Hofmeyr S, Somayaji A. The evolution of system-call monitoring. In: Proceedings of the 2008 annual computer

security applications conference, ACSAC '08. Washington DC, USA: IEEE Computer Society; 2008. p. 418–30.

Gilbert P, Chun B-G, Cox LP, Jung J. Vision: automated security validation of mobile apps at app markets. In: Proceedings of the second international workshop on mobile cloud computing and services, MCS'11. New York, NY, USA: ACM; 2011. p. 21–6. http://dx.doi.org/10.1145/1999732.1999740.

Guo C, Wang HJ, Zhu W. Smart-phone attacks and defenses. In: HotNets III; 2004.

Houmansadr A, Zonouz S, Berthier R. A cloud-based intrusion detection and response system for mobile phones. In: Dependable Systems and Networks Workshops (DSN-W); 2011. p. 31–2. IEEE/IFIP 41st international conference on IEEE, 2011.

Kaspersky mobile security, http://www.kaspersky.com/mobile_downloads/; 2011.

Kim H, Karp B. Autograph: toward automated, distributed worm signature detection. In: Proceedings of the 13th conference on USENIX security symposium. Berkeley, CA, USA: USENIX Association; 2004. p. 19. 19.

J. Kirk. New android malware texts premium-rate numbers. ComputerWorld.

Kreibich C, Crowcroft J. Honeycomb: creating intrusion detection signatures using honeypots. SIGCOMM CCR 2004;34:51–6.

Lange M, Liebergeld S, Lackorzynski A, Warg A, Peter M. L4android: a generic operating system framework for secure smartphones. In: Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices, SPSM '11. New York, NY, USA: ACM; 2011. p. 39–50. http://dx.doi.org/10.1145/2046614.2046623.

Leech M, Ganis M, Lee Y, Kuris R, Koblas D, Jones L. RFC 1928: SOCKS Protocol Version 5; Apr. 1996.

V. Library. <http://www.viruslibrary.com/>.

Liu L, Yan G, Zhang X, Chen S. Virusmeter: preventing your cellphone from spies. In: RAID. Springer; 2009. p. 244–64.

Miettinen M, Halonen P. Host-based intrusion detection for advanced mobile devices. Advance Information Networking and Applications 2006;2:72–6.

Oberheide J, Cooke E, Jahanian F. Cloudav: N-version antivirus in the network cloud. In: van Oorschot PC, editor. USENIX security symposium. USENIX Association; 2008a. p. 91–106.

Oberheide J, Veeraraghavan K, Cooke E, Flinn J, Jahanian F. Virtualized in-cloud security services for mobile devices. In: Proceedings of the first workshop on virtualization in mobile computing; 2008b. p. 31–5.

S. Perez. Over 50 DroidDream malware apps removed from android market http://www.readwriteweb.com.

Porras P, Saidi H, Yegneswaran V. An analysis of the iKeeB iPhone botnet. MobiSec 2010:141–52.

Portokalidis G, Homburg P, Anagnostakis K, Bos H. Paranoid android: versatile protection for smartphones. In: Proceedings of the 26th Annual Computer Security Applications Conference; 2010. p. 347–56.

Racic R. Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. In: SecureComm; 2006. p. 1–10.

Rageagainsthecage. <http://downloadsquad.switched.com/tag/rageagainstthecage/>(2010).

Roesch M. Snort: lightweight intrusion detection for networks. In: USENIX-LISA; 1999. p. 229–38.

Schlegel R, Zhang K, Zhou X, Intwala M, Kapadia A, Wang X. Soundminer: a stealthy and context-aware sound trojan for smartphones. In: NDSS; 2011.

Singh S, Estan C, Varghese G, Savage S. Automated worm fingerprinting. In: SOSP; 2004. p. 4.

The terminal-emulator application, https://market.android.com/details?id=jackpal.androidterm&hl=en; 2010.

Wotring B, Potter B, Ranum M, Wichmann R. Host integrity monitoring using Osiris and Samhain. Syngress Publishing; 2005.

Yap TS, Ewe HT. A mobile phone malicious software detection model with behavior checker. In: Web and communication technologies and Internet-related social issues, Vol. 3597; 2005. p. 57–65.

Zhao B, Xu Z, Chi C, Zhu S, Cao G. Mirroring smartphones for good: a feasibility study. In: Mobiquitous; 2010.

Zou CC, Gong W, Towsley D, Gao L. The monitoring and early detection of internet worms. IEEE/ACM Trans. Netw 2005;13:961–74.

Saman Zonouz is an Assistant Professor in the Electrical and Computer Engineering Department at the University of Miami. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2011. He has worked on intrusion response and recovery, information flow-based security metrics for power-grid critical infrastructures, online digital forensics analysis and monitorless recoverable applications. His research interests include: computer security and survivable systems, control/game theory, intrusion response and recovery systems, automated intrusion forensics analysis, information flow analysis-based security metrics, intrusion detection and correlation, and trustworthy power-grid critical infrastructures.

Amir Houmansadr is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, advised by Prof. Nikita Borisov within the Hatswitch research group. Amir earned his M.Sc. and B.Sc. degrees from the Electrical Engineering Department at the Sharif University of Technology in 2005 and 2003, respectively. His research revolves around various network security and privacy problems. Amir has been involved in research projects, including stealthy anonymous communications, social network botnets, covert timing channels, and smartphone security.

Robin Berthier is a research scientist at the University of Illinois at Urbana-Champaign, working with Prof. William H. Sanders in the Information Trust Institute. Robin graduated from the Reliability Engineering Department at the University of Maryland in 2009. His doctoral dissertation introduced a new architecture to increase the scalability of high-interaction honeypots, and combined network datasets of different granularities to offer unique attack forensics capabilities to security analysts. His current research interests include advanced intrusion detection systems and the security of critical infrastructures.

Nikita Borisov is an assistant professor of Electrical and Computer Engineering at the University of Illinois. His research interests are online privacy and Internet-scale distributed systems. He is the co-designer of the "off-the-record" (OTR) instant messaging protocol and was responsible for the first public analysis of 802.11 security. Prof. Borisov received his PhD from the University of California, Berkeley in 2005 and a BMath from the University of Waterloo in 1998.

William H. Sanders is a Donald Biggar Willett Professor of Engineering and the Director of the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. He is a Fellow of the IEEE and the ACM, a past Chair of the IEEE Technical Committee on Fault-Tolerant Computing, and past Vice-Chair of the IFIP Working Group 10.4 on Dependable Computing. He was the founding Director of the Information Trust Institute (www.iti.illinois.edu) at Illinois. Dr. Sanders's research interests include secure and dependable computing and security and dependability metrics and evaluation, with a focus on critical infrastructures.