ISSN No: 2454-9614

A Novel Approach for Obfuscated Malware Analysis on Smartphone

P. Sathyabama, G. Kirubhakar [*]

Department Of Computer Science and Engineering, Surya Engineering College,
Erode-638107, Tamilnadu, India.

*Corresponding Author: P. Sathyabama

E-mail: sathyaprabakar007@gmail.com,

Received: 14/11/2015, Revised: 16/12/2015 and Accepted: 05/03/2015

Abstract

The rapid growth of Smartphone sales has come hand in hand with a similar increase in the number and sophistication of malicious software targeting these platforms. Malware analysis is a thriving research area with a substantial amount of still unsolved problems. A major source of security problems is precisely the ability to incorporate third-party applications from available online markets. In the case of smart phones, the impressive growth both in malware and begin apps is making increasingly unaffordable any human-driven analysis of potentially dangerous apps. Malware samples consists of hiding and obfuscating modules containing malicious functionality in places that static analysis tools overlook ALTERDROID, is a open source tool for detecting, through reverse engineering, obfuscated functionality in components distributed as parts of an app package. Such components are often part of a malicious app and are hidden outside its main code components, as code components may be subject to static analysis by market operators. The key idea in ALTERDROID consists of analyzing the behavioural differences between the original app and an altered version where a number of modifications. The Malware applications are shown in the screen, and then the user can uninstall the malicious application. The experimental results obtained by testing ALTERDROID over relevant apps and malware samples support the quality and viability of our proposal.

Keywords: Droidkungfu, malicious code, ginger master

Reviewed by* **ICETSET'16 *organizing committee*

1. Introduction

Smartphone is quickly becoming the dominant device for accessing Internet resources. Sales of smart phones overtook PC sales in the global market in 2010. Shipments of smart phones surpassed those of feature phones in Western Europe in 2011. According to May 2011 Nielsen survey, smart phones outsold feature phones in the US in this same period. Compared to 5.9 billion worldwide mobile phone subscribers, Smartphone usage (835 million) is still steadily increasing. IDC predicts Smartphone shipments will approach one billion in 2015. Smart phones offer many more functions than traditional mobile phones. In addition to a preinstalled mobile operating system, such as IOS, Android, or Windows Mobile, most smart phones also typically support carrier networks, Wi-Fi connectivity, and Bluetooth so that users can access the Internet to download and run various third party applications. Most Smartphone support Multimedia Message Service (MMS) and include embedded sensors such as GPS, gyroscopes, and accelerometers, as well as a high-resolution camera, a microphone, and a speaker.

Smartphone's increasing popularity raises many security concerns. Their central data management makes them easy targets for hackers. Since the first mobile phone viruses emerged in 2004, Smartphone users have reported significant malware attacks. In the last seven months of Because of their unique characteristics, 2011, malware attacks on the Android platform increased 3,325 percent. As the use of Smartphone continues its rapid growth, subscribers must be assured that the services they offer are reliable, secure, and trustworthy. In a Smartphone threat model, a malicious user publishes malware disguised as a normal application through an app store or website. Users will unintentionally download the malware to a Smartphone, which carries a large amount of sensitive data.

After infiltrating a Smartphone, the malware attempts to control its resources, collect data, or redirect the Smartphone to a premium account or malicious website. This model divides a Smartphone into three layers:

- *Application layer* includes all of the Smartphone's apps, such as social networking software, email, text messaging, and synchronization software.
- *Communication layer* includes the carrier networks, Wi-Fi connectivity, Bluetooth network, Micro USB ports, and Micro SD slots. Malware can spread through any of these channels.
- *Resource layer* includes the flash memory, camera, microphone, and sensors within a Smartphone. Because smart phones contain sensitive data, malware targets their resources to control them and manipulate data from them.

An attack forms a loop starting with the launch of the malware, moving through the Smartphone's application, communication, and resource layers, on to premium accounts/malicious websites, and back to the malicious user shows such an attack.

Smartphone also feature high-quality audio and video recording capabilities. Sensitive pieces of information that can be captured by these devices could be easily leaked by malware residing on the Smartphone. Even apparently harmless capabilities have swiftly turned into a potential menace. For example **[1],** access to the accelerometer or the gyroscope can be used to infer the location of screen taps and, therefore, to guess what the user is typing (e.g., passwords or message contents).

Similarly, the Radio Data System (RDS) embedded in most AM/FM channels can be exploited to inject attacks on Software Defined Radio (SDR) systems. A major source of security problems is precisely the ability to incorporate third-party applications from available online markets. Thus, security measures at the market level constitute a primary line of defence. Many market operators carry out a revision process over submitted apps that involve some form of security testing. Official details about such revisions remain unknown, but the constant presence of malware in many markets and recent research studies suggest that operators cannot afford to perform an exhaustive analysis over each app submitted for release to the general public. This is further complicated by the fact that determining which applications are malicious and which are not is still a formidable challenge, particularly for the so-called gray ware namely, apps that are not fully malicious but that constitute a threat to the user security and privacy.

1.1 MALWARE

Smartphone malware falls into three main categories: viruses, Trojans, and spyware. Viruses are typically disguised as a game, security patch, or other desirable application, which a user downloads to a Smartphone. Viruses can also spread through Bluetooth. Two Bluetooth viruses have been reported in smartphones:

- **Blue jacking** sends unsolicited messages over Bluetooth to a Bluetooth-enabled device within a limited range (usually around 33 feet).
- **Blue snarfing** accesses unauthorized information in a smartphone through a Bluetooth connection.

Most smartphone Trojans are related to activities such as recording calls, instant messaging, finding a location via GPS, or forwarding call logs and other vital data. According to **[6],** Smart Message System Trojans comprise a large category of mobile malware that run in an application’s background and send SMS messages to a premium rate account owned by an attacker. Hippo SMS, for example, increases user’s phone charges by sending SMS messages to premium mobile accounts and blocks service provider’s messages alerting users of the additional charges.

Spyware collects information about users without their knowledge. According to a 2011 report, spyware was the dominant malware affecting Android phones, accounting for 63 percent of the samples identified.

1.2 Mobile Threat Model

Types of Threat In mobile threat model include main two types of threats: gray ware, and Anti-spyware. It distinguish between the three predicated on their distribution method, lucidity, and notice to utilize. The main focuses especially on malware; personal spyware and gray ware use different attack vectors, have different motivations, and require different bulwark mechanisms.

1.2.1 Gray ware:

Gray ware refers to a malignant software or code that is considered to fall in the "grey area" between mundane software and a virus. Gray ware **[7]** is a term for which all other maleficent or exasperating software such as adware, spyware, track ware, and other maleficent code and malevolent shareware fall under.

1.2.2 Anti-spyware

Anti-spyware is a type of software that is designed to detect and abstract unwanted spyware programs.

Spyware is a type of malware that is installed on a computer without the utilizer's cognizance in order to amass information about them. This can pose a security risk to the utilize, but more frequently spyware degrades system performance by taking up processing puissance, installing supplemental software, or redirecting users' browser activity.

1.2.3 Obfuscated Smartphone Malware

Smartphone had the impressive growth both in malware and benign apps are making increasingly unaffordable any human-driven analysis of potentially dangerous apps. This has consolidated the need for intelligent analysis techniques to aid malware analysts in their daily functions. Furthermore, Smartphone malware is becoming increasingly stealthy and recent specimens are relying on advanced code obfuscation techniques to evade detection by security analysts. For instance, Droid KungFu has been one of the major Android malware outbreaks. It started on June 2011 and has already at least six known different variants. It has been mostly distributed through official or alternative markets by piggybacking the malicious payload into a variety of legitimate applications. Such a payload is encrypted into the app’s assets folder and decrypted at runtime using a key stored in a local variable and located at one class.

Another remarkable example is Ginger Master, the first malware using root exploits for privilege escalation on Android 2.3. The main payload was stored as PNG and JPEG pictures in the assets file, which were interpreted as code once loaded by a small hook within the app. More sophisticated obfuscation techniques, particularly in code, are starting to materialize (e.g., stego malware).

These techniques and trends create an additional obstacle to malware analysts, who see their task further complicated and have to ultimately rely on carefully controlled dynamic analysis techniques to detect the presence of potentially dangerous pieces of code.

1.2.4 Fault Injection:

Fault injection is a technique for improving the coverage of a test by introducing faults to test code paths, in particular error handling code paths that might otherwise rarely be followed. It is often used with stress testing and is widely considered to be an important part of developing robust software. Robustness testing (also known as Syntax Testing, Fuzzing or Fuzz testing) is a type of fault injection commonly used to test for vulnerabilities in communication interfaces such as protocols, command line parameters, or APIs.

The propagation of a fault through to an observable failure follows a well defined cycle. When executed, a fault may cause an error, which is an invalid state within a system boundary **[8].** An error may cause further errors within the system boundary, therefore each new error acts as a fault, or it may propagate to the system boundary and be observable. When error states are observed at the system boundary they are termed failures. This mechanism is termed the fault-error-failure cycle and is a key mechanism in dependability.

2. Overview

ALTERDROID, an open source tool for detecting, through reverse engineering, obfuscated functionality in components distributed as parts of an app package. Such components are often part of a malicious app and are hidden outside its main code components (e.g. within data objects), as code components may be subject to static analysis by market operators. The key idea in ALTERDROID consists of analyzing the behavioural differences between the original app and an altered version where a number of modifications (faults) have been carefully introduced.

Such modifications are designed to have no observable effect on the app execution, provided that the altered component is actually what it should be (i.e., it does not hide any unwanted functionality). For example, replacing the value of some pixels in a picture or a few characters in a string encoding an error message should not affect the execution. However, if after doing so it is observed that a dynamic class loading action crashes or a network connection does not take place, it may well be that the picture was actually a piece of code or the string a network address or a URL.

At high level, ALTERDROID has two differentiated major components: fault injection and differential analysis **[4].** The first one takes a candidate app—the entire package—as input and generates a fault-injected one. This is done by first extracting all components in the app and then identifying those suspicious of containing obfuscated functionality. Such identification is done on an anomaly-detection basis by comparing specific statistical features of the component's contents with a predefined model for each possible type of resource (i.e., code, pictures and video, text files, databases, etc.).

Faults are then injected into candidate components, which are subsequently repackaged, together with the unaltered ones, into a new app **[8].** This process admits simultaneous injection of different faults into different components and it is driven by a search algorithm that attempts to identify where the obfuscated functionality is hidden.

Both the original and the fault-injected apps are then executed under identical conditions (i.e., context and user inputs), and their behaviour is monitored and recorded in the form of two behavioural signatures.

Such signatures are merely sequential traces of the activities executed by the app, such as for example opening a network connection, sending or receiving data, loading a dynamic component, sending an SMS, interacting with the file system, etc. Both behavioural signatures are then treated as in a string-to-string correction problem, in such a way that computing the Levenshtein (edit) distance between them returns the list of observable differences in terms of insertions, deletions, and substitutions.

Such a list, called the differential signature, is finally matched against a rule-set where each rule encodes a relationship between the type of presumably hidden functionality and certain patterns in the differential signature.

The functional components of ALTERDROID, a prototype implementation of our differential fault analysis model for Android apps. The system includes instantiations for key tasks such as identifying components to be fault-injected and a search-based approach to track down obfuscated components in an app.

ALTERDROID's functional architecture supports distributed deployment of different modules, which allows running various analysis tasks in parallel and also potentially offloading them to the cloud. Differential fault analysis for detecting obfuscated malware functionality in smartphone apps. The models for fault injection operators, behavioural signatures and rule-based analysis of differential behaviour are described.

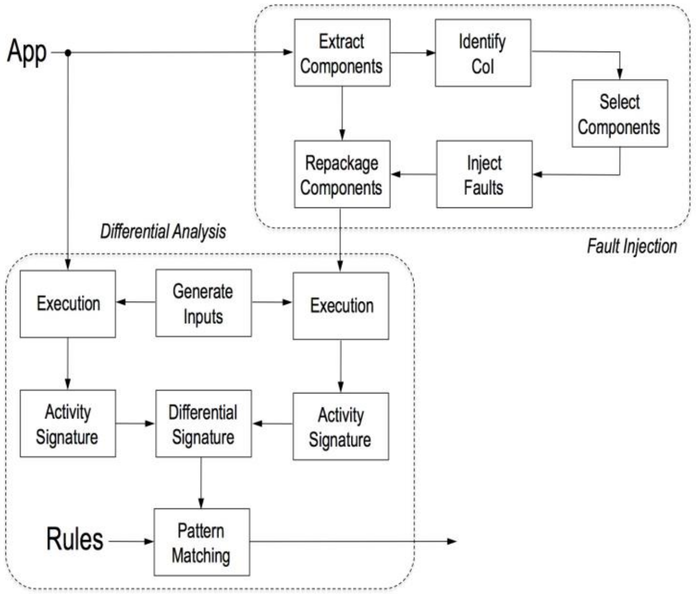

Figure 2.1 Basic Architecture of Alter droid

2.1 Malware Samples

Android malware samples that incorporate hidden functionality in repackaged apps: Droid KungFu, AnserverBot, and Ginger Master.

2.1.1 Droid KungFu (DKF)

DKF's main goal is to collect details about the infected Android device, including the IMEI (International Mobile Station Equipment Identity) number, phone model, and OS version **[3].** It is mostly distributed through open or alternative markets via repackaging that is, by piggybacking the malicious payload into various legitimate applications. Apps infected with DKF are distributed together with a root exploit hidden within the app's assets, namely, Rage against the Cage (RAC). To hinder static analysis, this encrypted payload is only decrypted at runtime.

*2.1.2 Ginger Master (GM***)**

GM is the first known malware to use root exploits for privilege escalation on Android 2.3. Its main goal is to exfilt rate private information such as the device ID (IMEI number, MSI number and so on) or the contact list stored in the phone. GM is generally repackaged with a root exploit known as Ginger Break, which is stored as a PNG and a JPEG asset file. Right after infecting the device, GM connects to the C&C server and fetches new payloads.

3. Related work

A new behaviour-based anomaly detection system is used to detecting meaningful deviations in a mobile application's network behaviour. The main goal of the proposed system is to protect mobile device users and cellular infrastructure companies from malicious applications by:

- Identification of malicious attacks or masquerading applications installed on a mobile device, and
- Identification of republished popular applications injected with a malicious code (i.e., repackaging).

More specifically, it attempts to detect a new type of mobile malware with self-updating capabilities that were recently found on the official Google Android marketplace.

Mobile devices and their application marketplaces drive the entire economy of the today's mobile landscape. Android platforms alone have produced staggering revenues, exceeding five billion USD, which has attracted cybercriminals and increased malware in Android markets at an alarming rate. To better understand this slew of threats, it presents Copper Droid, an automatic VMI-based dynamic analysis system to reconstruct the behaviours of Android malware. The novelty of Copper Droid lies in its agnostic approach to identify interesting OS- and high-level Android-specific behaviours. It reconstructs these behaviours by observing and dissecting system calls and, therefore, is resistant to the multitude of alterations the Android runtime is subjected to over its life-cycle.

Android mobile devices are enjoying a lion's market share in smart phones and mobile devices. This also attracts malware writers to target the Android platform. Recently, a new Android malware distribution channel: releasing malicious firmware's with pre-installed malware to the wild. This poses significant risk since users of mobile devices cannot change the content of the malicious firmware.

4. Proposed Approaches

In this paper we describe ALTERDROID, a tool for detecting, through reverse engineering, obfuscated functionality in components distributed as parts of an app package. Such components are often part of a malicious app and are hidden outside its main code components (e.g. within data objects), as code components may be subject to static analysis by market operators. The key idea in ALTERDROID consists of analyzing the behavioural differences between the original app and an altered version where a number of modifications (faults) have been carefully introduced. Such modifications are designed to have no observable effect on the app execution, provided that the altered component is actually what it should be (i.e., it does not hide any unwanted functionality).

For example, replacing the value of some pixels in a picture or a few characters in a string encoding an error message should not affect the execution. However, if after doing so it is observed that a dynamic class loading action crashes or a network connection does not take place, it may well be that the picture was actually a piece of code or the string a network address or a URL.

It Performs,

- Inject faults into apps;
- Represent behavioural differences between apps;
- Deduce properties from such behavioural differences considering injected faults and observed differences.
- Framing the rules to detect the malware

Advantages:

- ALTERDROID is designed and built to allow ease of tailoring and flexibility in functionality
- It provides powerful model for fault injection operators, behavioural signatures and rule based analysis of different behaviour.

4.1 Analysis steps

4.1.1 Classification on installed Apps in Mobile Phone

ALTERDROID is an open source tool for creating obfuscated functionality in components distributed as parts of an app package. It consists of analyzing the behavioural differences between the original app and an altered version where a number of modifications (faults).In this module, first creates ALTERDROID tool for malware detection. Next it first classifies the installed apps in mobile phone. Classified apps such as predefined app, system app and plays tore app.

4.1.2 Explore Application Manifest

Applications are identified with file extension “APK”. Each APK package runs in its own Environment. The process ownership is identified with the APK id in the manifest of the file application. The manifest file is called “AndroidManifest.xml” and is located in the application’s root directory.

The contents of manifest file identify components, classes, services, access rights etc. In this module it store working procedure and original behaviour of the app.

4.1.3 Detect Application Enabled Permissions

The permissions required by the application to access components and services in Android Environment **[5].** The permission offered by the application to allow access to its components and services. It allocates the permission to the app and disables the permission to the original app. Malware can be detected based on these methods

ALTERDROID monitors the execution of different activities:

- Crypto: generated when calls to the cryptographic API are invoked;
- Net-open, net-read, net-write: associated with network I/O activities (opening a connection, receiving, and sending data);
- File-open, file-read, file-write: associated with file system I/O activities (opening, reading, and writing);
- SMS, call: generated whenever a text message or a phone call is sent or received;
- Leak: generated whenever the app leaks private information, as determined by Taint droid; and
- DEX load: generated when an app loads native code.

4.1.4 Remove or Uninstall malicious apps

Android malware samples that incorporate hidden functionality in repackaged apps: Droid KungFu, Anserver Bot, and Ginger Master. Ginger Master, the first malware using root exploits for privilege escalation on Android 2.3. The main payload was stored as PNG and JPEG pictures in the assets file, which were interpreted as code once loaded by a small hook within the app .In this module it uninstall or remove the malicious apps.

5. Evaluation

We next report a number of experimental results obtained with our prototype implementation of ALTERDROID. These results illustrate how our system can be used by market operators and security analysts to facilitate the analysis of complex obfuscated mobile malware. We first present the results of testing ALTERDROID against two datasets of Smartphone malware samples found in the wild, including a performance analysis of the entire differential fault analysis process. We finally discuss in more detail three representative case studies.

5.1 Other Recent Specimens:

We have analyzed some of the most recent specimens hitting both official and unofficial markets. Although obfuscation techniques and algorithms might vary, results confirm that malware keeps hiding payloads within app resources such as images or XML files. The most significant analyzed specimens were:

Emmental: this malware sample targets users of several banks worldwide, collecting one-time passwords used to authorize transactions. Apps infected with Emmental are distributed together with an initial configuration containing a phone number where certain SMSs are sent and several Command and Control (C&C) URLs. To hinder static analysis, this configuration is only decrypted at runtime using Blowfish. According to a report from Trend Micro, Emmental was still active as of 2014.

Gamex: this specimen introduces an update component that enables it to retrieve new payloads, at runtime, from a C&C server. Its main goal is to exfiltrate private information such as the device ID (IMEI number, MSI number, and so on). Gamex obfuscates the main payload using XOR operations while hiding it into the app resources—specifically, a file called logos.png.

SmsSpy: this malware is similar to Emmental in terms of sophistication and distribution strategy . It also uses Blowfish to encrypt its payload and hinder analysis. The payload is generally stored in a file called data.xml and the decryption key is hardcoded in the app code.

6. Conclusion and Future Enhancements

In this work ALTERDROID tool is used to identify the malware analysis based on the differential analysis. Differential fault analysis in the way implemented by ALTERDROID is a powerful and novel dynamic analysis technique that can identify potentially malicious components hidden within an app package. Additionally, empowering dynamic analysis with a fault injection approach can be used to differentiate “gray” from legitimate behaviour when analyzing gray ware. This is a good complement to static analysis tools, more focused on inspecting code components but possibly missing pieces of code hidden in data objects or just obfuscated. Finally, we believe that differential fault analysis is an effective technique to detect stego malware—malware using advanced hiding methods such as steganography.

6.1 future works

As future work, we are currently extending ALTERDROID to support differential fault analysis over distinguishable components such as those involving Dex byte code. ALTERDROID open source prototype with a versatile design that can be the basis for further research in this area.

References

[1] Shabtai, L.Tenenboim-Chekina, D.Mimran, L.Rokach, B.Shapira, Y.Elovici., ‘‘Mobile malware detection through analysis of deviations in application network behavior”., Department of Information Systems Engineering,2014.

[2] Kimberly Tam, Salahuddin J. Khan , Aristide Fattori, and Lorenzo Cavallaro., “CopperDroid: Automatic Reconstruction of Android Malware Behaviors”, Systems Security Research Lab, Royal Holloway University of London,2015.

[3] Min Zheng, Mingshen Sun, John C.S. Lui., “DroidRay: A Security Evaluation System for Customized Android Firmwares”,Computer Science & Engineering Department The Chinese University of Hong Kong,2014.

[4] G. Suarez-Tangil, F. Lombardi, J. E. Tapiador, and R. Di Pietro, “Thwarting obfuscated malware via differential fault analysis”, IEEE Computer, vol. 47, no. 6, pp. 24–31, June 2014.

[5] Desnos and et al., “Androguard: Reverse engineering, malware and goodware analysis of android applications”, https://code.google.com/p/androguard/, Visited Feb, 2015.

[6] G. Suarez-Tangil, J. E. Tapiador, P. Peris, and A. Ribagorda, “Evolution, detection and analysis of malware for smart devices”, IEEE Comms. Surveys & Tut., vol. 16, no. 2, pp. 961–987, May 2014.

[7] M. Rangwala, P. Zhang, X. Zou, and F. Li, “A taxonomy of privilege escalation attacks in android applications”, Int. J. Secur. Net., vol. 9, no. 1, pp. 40–55, Feb 2014.

[8] L. K. Yan and H. Yin, “Droidscope: seamlessly reconstructing the os and Dalvik semantic views for dynamic Android malware analysis,” in Proc. USENIX, ser. Security’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 29–29.

[9] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, “Dendroid: A text mining approach to analyzing and classifying code structures in android malware families,” Expert Systems with Applications, vol. 41, no. 1, pp. 1104–1117, 2014.

[10] V. I. Levenshtein, “Binary Codes Capable of Correcting Deletions, Insertions and Reversals,” S. Physics Doklady, vol. 10, p. 707, 1966.

[11] T. Kumazawa and T. Tamai, “Counter example-based error localization of behavior models,” in Proc., ser. NFM’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 222–236.

[12] G. Suarez-Tangil, F. Lombardi, J. E. Tapiador, and R. Di Pietro, “Thwarting obfuscated malware via differential fault analysis,” IEEE Computer, vol. 47, no. 6, pp. 24–31, June 2014.

[13] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, “Smartdroid: an automatic system for revealing UI-based trigger conditions in Android applications,” in Proc. ACM, ser. SPSM ’12. New York, NY, USA: ACM, 2012, pp. 93–104.

[14] Android, “Android developers,” Visited Feb. 2015, http://developer.android.com/.