Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 78 (2016) 587 – 594

Procedia Computer Science

International Conference on Information Security & Privacy (ICISP2015), 11-12 December 2015, Nagpur, INDIA

# Anti-Hijack: Runtime Detection of Malware Initiated Hijacking in Android

Venkatesh Gauri Shankar[a,*], Gaurav Somani[a]

[a]Department of Computer Science and Engineering, Central University of Rajasthan, Ajmer, Rajasthan, India

## Abstract

According to studies, Android is having the highest market share in smartphone operating systems. The number of Android apps (i.e. applications) are increasing day by day. Consequent threats and attacks on Android are also rising. There are a large number of apps which bypass users by hiding their functionalities and send users sensitive information and data across the network. Due to flexibility and openness of Android operating system, attack surfaces are being introduced every other day.

In this paper, we are addressing detection of two fatal malware attacks; intent based hijacking and authenticated session hijacking. We have used the concept of honey-pot in detection of these two authentication hijacking problems. In order to achieve this, we have tested various apps and their interaction with the honey-pot maintained by real device or an emulator. We have designed benign app as a honey framed app. We argue that hijacking malware can be detected with higher accuracy using our method at run-time as compared to the traditional machine learning methods. Our approach, Anti-Hijack, which has provided the detection accuracy as high as 96%. This has been highly accurate to detect the unwanted interaction between hijacking malware and designed benign app. We have tested our approach on a strong data-set of Android apps for experiment and identifying vulnerable points. Our detection method Anti-Hijack is a novel contribution in this area which provides light weight, device operated run-time detection at hijacking malware.

Keywords: Android malware; Anti-Hijack; intent; vulnerabilities; smartphone; security; honey-pot.

## 1. Introduction

Mobile operating system, like the desktop operating system has a major impact in the operating system market [16]. Popular and largest market share holder operating systems in the smartphone market are Android and iOS. Google Android was released as a flexible and open source operating system being adapted by many hardware vendors. It covers approximate 90% market share in Q4 of 2014 [18]. Popularity of Android operating system is due to many features like access to wide variety of Personally Identifiable Information (PII), wide array of hardware, application integrated Google services, good user widget, large number of apps in many app stores, flexible and also openness in availability [17, 18]. As soon as number of apps and activation of Android devices are increasing parallel attacks and vulnerable points are also increasing [14].

* Corresponding author. Tel.: +91-8769196918 ; fax: +0-000-000
E-mail address: venkateshgaurishankar@gmail.com

doi:10.1016/j.procs.2016.02.105

Vulnerabilities are weak points which reduce systems information security assurance. It is an intersection of three linked element which are system flaws, attack access to system flaws and attacker capability to exploit the flaw. When an attacker uses application tool or techniques then this vulnerable point is used as an attack surface [24].

## 2. Related Work

Chin, Erika et al. [23] analyse Android apps execution and identify hijack risks in apps components. They develop a tool ComDroid that identifies app's communication vulnerability. Their analysis provides that developer usually use the robust field of a component like logic instead of explicit logic of the intent, which has the unusual disadvantages of making intent public. Kelly Casteel,Owen Derby, Dennis Wilson[27] build a working exploit which takes advantages of intent based vulnerability & develop a statically arranged Android analysis framework to impose an app for malign content infected parts. Sebastian H¨obarth and Rene Mayrhofer provide additional access to application permanent privilege escalation is required. They present a framework that can uses arbitrary temporary exploits on Android de-vices to achieve permanent root capabilities.Roee Hay [26] presents a new found vulnerability in Android which breaks its sandbox process. This vulnerability uses many apps also system intent apps. Adam Cozzette [24] focuses on Intent spoofing attacks, by taking an attack scenario where a vulnerable application has components which only expects to receive intents from other component of same application. Okolie C.C et al. [20] maintain glide checking of Android smartphones with the help of an app process maturation to change some configure methods for compiling data and vulnerable intent. In this paper, a phenomenon is given to test and indicating the strong network of the Android using the new slide testing and hijacking tool related to Linux. The result identifies that there is an identification in the security range of the different Android device versions but version 4.2 is more robust than the others. Mansfield-Devine, Steve [29] describe the week points of Android Architecture by using MWR's exploitation framework mercury. They analyse cross application exploitation, Inter app communication and SQL-Injection vulnerability with the existing modules of mercury framework. Enck, William and Gilbert et al. [25] maintain how third-party app's uses their private data for processing. They uses a new concept as TaintDroid, an e cient system-range dynamic taint detection and analysis system providing of simultaneously identifying multiple sources of user's sensitive data. TaintDroid provides
realtime analysis by using Android virtualized execution environment. T.BIeasing et al. [19] present an Android app sand-boxing technique which is used to find static as well as dynamic analysis with Android apps to ineluctably notice malign app's. Static analysis is used to find the app's for malign activity with no runnable instance. Dynamic anal-ysis is used to install an app in a block of isolated environment which is sandbox. Sandbox performs and maintains app identification with the Android device for execution. Cerbo et al. [21] demonstrate logic for smartphone activity based execution, to notice malign app's which are hiding their activities. This logic is mainly imposed with the Android device and applies on its secure component specially permissions in each app's. Je Lessard and Gary C. Kessler [28] present an experiment in acquiring information from an Android device using multiple methods dd analysis with FTK, creating a dd image of memory, examination of Memory, recovered documents, logical examination, logical analysis of specific databases, data extraction with the CelleBrite UFED.

We have designed a runtime malware detection framework named Anti-Hijack which is working on two fatal malware attacks related to intent and session. We have used the technique of honey-pot based environment to identify session and intent based malware. The accuracy of Anti-Hijack is as high as 96% in Subsection 6.1. Anti-Hijack generates less false positive because no legitimate or benign application uses honey-pot log file. We have measured efficiency of Anti-Hijack using tenfold cross validation after calculating precision, recall and H-mean in Subsection 6.2. The statistical comparison of Anti-Hijack with other existing related work is in Table 1.

| Framework/Tools | Accuracy (in %) | H-Mean |
|---|---|---|
| ComDroid | 70 | - |
| Kelly et al. | 80 | - |
| SandBox | 85 | .783 |
| Adam et al. | 84 | - |
| TaintDroid | 90 | .801 |
| Cerbo et al. | 86 | .79 |
| Je et al. | 81 | - |
| Anti-Hijack | 96 | .834 |

Table 1: Statistical comparison of Anti-Hijack with existing related work

## 3. Intent and Session Based Vulnerabilities

### 3.1. Intent Based Vulnerabilities

Intents are the mechanism part for the parallel communication between components of Android[3]. Intents are the special feature used to bind services, passing notification to broadcast receiver and to start activities and services. In Android, a component may receive intents from the other parts in the same Android app. Attackers can configure the attribute of manifest file android: exported which allows to accept intent from outside [4]. One more security concern is intent interception. It allows a malicious app to receive an intent that was not intended for it. Due to the intent interception, there is a high chance of sensitive data leakage. For example, if a malign activity intercepted an intent then it would appear on the screen as a benign or legitimate activity[24]. Following are the three different types of attacks, which come under intent based hijacking.

#### 3.1.1. Activity Hijacking

Activity hijacking occurs when an intent is sent out implicitly and malicious attacker wants to filter that intent. For example, in an app, where user searches for an item and the app opens up the search results in a browser. The hijack attacker will intercept the URL before passing the intent to the real browser. The attacker is now capable of tracking user activity and attaches any data with the URL[24].

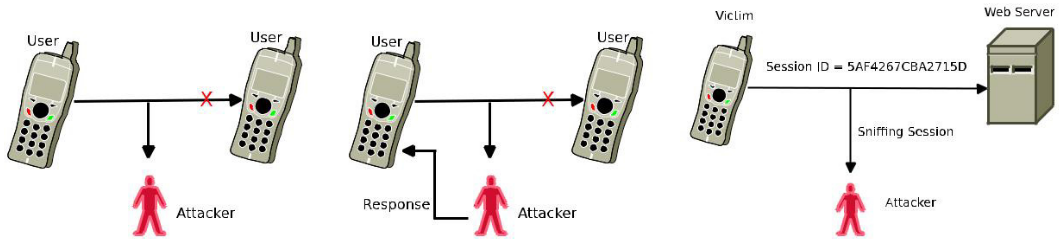
#### 3.1.2. Service Hijacking

Service hijacking happens when multiple services can handle only one intent then the Android operating system resolves the destination, not by prompting user but based on which app was installed first. It is an option for attacker to get a malicious app on to the smartphone before the benign or legitimate app. Attacker can access data, spoof results and also hijack the whole session[24].

#### 3.1.3. Broadcast Hijacking

Broadcast can easily be captured by malicious components because it does not require receiver signature. On other hand, due to not setting receiver signature priority, manifest file can easily be manipulated by malicious software. Attacker can access data and produces denial of service [24].

Intent hijacking tactics is shown in Fig. 1(a) without response and Fig. 1(b) shows intent hijacking tactics with attacker response.



(a) Intent hijacking tactics (b) Intent hijacking with response (c) Session hijacking tactics

Fig. 1: Session and intent hijacking

### 3.2. Session Hijacking Vulnerabilities

In session based hijacking, attacker can filtrate, command and control the whole user defined targeted device. In such type of hijack, a malign activity is imported in place of legitimate activity. The malign activity registers to get an app's implicit intents and started in place of designated activity. When activity hijack is successful then the user component may be opened to a secondary false response attack and activity attacker can return malign response such as valuable information of device to invoking app. Fig. 1(c) shows the activity hijacking tactics.

### 3.3. New Malware Family Related to Session and Intent Vulnerability

The importance of out work become high, as there is a significant increase in intent/session hijacking based mal-ware recently. In Q3 and Q4 of year 2014, Android malware[22] samples related to the intent based vulnerability and session hijacking have been identified as new malware families in Table 2. The intensity of malware is identified as low, high or risky. Risky intensity malware can be harmful to the user for some time.

| Intent Based Family | Session Based Family | Intensity |
|---|---|---|
| FakeBank | DroidSleep | Low |
| VMWol | FaceNi | High |
| FakeJobO er | FakeAV | Risky |
| CarBerp | FakeFlash | High |
| Cawitt | FakeMarket | Risky |
| Cosha | Obad | High |
| CruseWind | DroidDeluxe | High |

Table 2: Current intent and session hijack based Android malware

## 4. Honey-pot and Proposed Methodology

### 4.1. Honey-pot and Honey-net

Honey-pot is an information collecting and suspicious measurement framework which captures anything with maliciousness and new tools for identifying attack too. It is used in network unit as a controlled network where every packet entering and leaving is monitored, captured and analysed. It is also the phenomenon of detection, protection and defence. This concept can be applied to mobile system network to catch suspicious malign mobile users. Honey-pots are basically two types. First one is physical honey-pot which is actual computer or mobile device that are set-up with more logging and security features. On the other hand, virtual honey-pots are a software package that allows you to fake many computer or mobile devices distributions at various places over the network from one mobile device or computer. Honey-net is the collection of more than one honey-pot on the mobile device or computer. The main scenario of honey-pot is in Fig. 2.

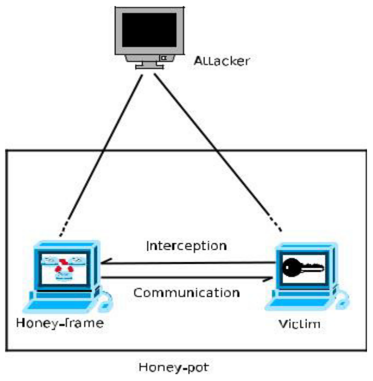


Fig. 2: Honey-pot methodology

### 4.2. Proposed Methodology

As it is clear in Section 3 that intent and session hijack attacks benign app by trapping their intent and session. Therefore an enhancement of honey-pot methodology fit here. In our scenario, honey-pots are real devices or an emulated environment which is used to monitor intercepted system call between test app and our specially designed honey framed benign app. All the interactions are stored in a log. This log file (SysCall) is used for monitoring, detecting and analysing vulnerable Android apps in complete functionalities of Anti-Hijack. As shown in Fig. 3, we have deployed single system as honey-pot. We have started monkey runner tool to record system call separately for tested app and designed benign app.

#### 4.2.1. Android Based SDK Tools Used [5]

ADB[2]: We have used ADB which is a robust tool that communicate with an Android emulator or Android virtual device. It contains five components:

Client: This runs on our developer machine. We can search client from a shell by using an adb command.

Server: This executes as a hide process on our developer machine. The server basically creates and control interaction between the client and the adb daemon running on Android virtual device.

Daemon: This runs as a back ground on each emulator or Android device instances.

DumpSys[1]: It runs on the device and dumps fascinating information about the status of system services. Monkey Runner[13]: This tool is used for tracing system call of the tested app as well as benign app at runtime.

### 4.2.2. Experimental Setup

After installing SDK, we have deployed test app, honey framed benign app, monkey runner tool, targeted android API in a single form. All these makes a system used as honey-pot.

After deploying single system as honey-pot, we have started monkey runner tool to record system call separately for tested app and benign app. At this time this system behaves as a stand alone isolated system.

After recording system call, we have measured any interception between tested app and honey framed benign app. We found any interception in the form of intent or session based information flow leakage in the SysCall. In SysCall, this will be immediately detected by Anti-Hijack.

If there is no suspicious part in SysCall log file means app is benign. On the other hand if any suspicious pat is recorded then app is malicious in nature. Proposed framework is shown in Fig. 3.

Intent based and session based novel algorithms are given below in Algorithm. 1 and Algorithm. 2.

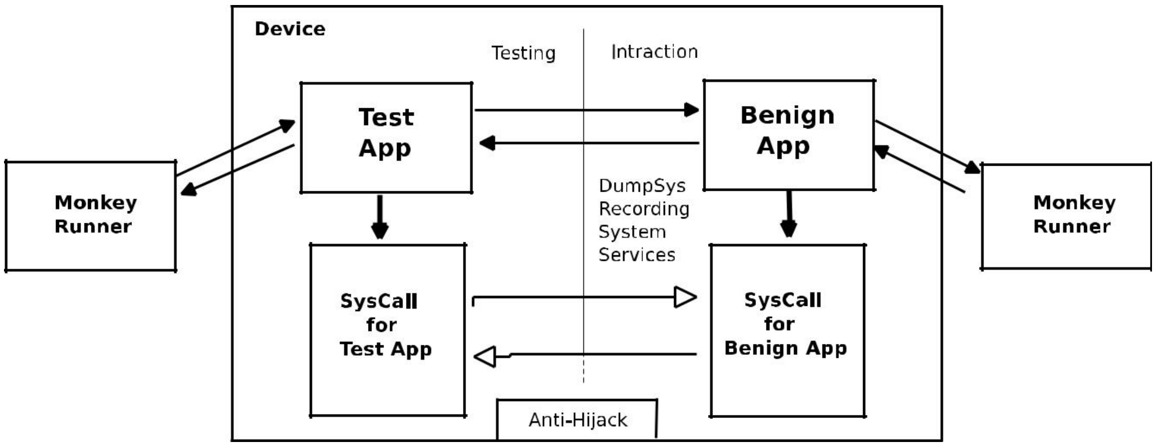


Fig. 3: Anti-Hijack: Proposed honey-pot framework for Android apps

## 4.3. Proposed Algorithm

```
Data: Tested App, Benign App
Result: Malicious or benign app
initialization: Android Virtual Device, Monkey;
while App is executed with emulated or real environment do
    Execute tested app and Benign app;
    Start monkey runner tool;
end
while Read SysCall do
    Monitor benign and Analysing tested app;
    Intent or session attack detected;
    if (activity or Services or IntentFltr S ysCall)then
        Monitor benign and Analysing tested app; Intent
        or session attack detected;
        Tested app is “malign“;
    else
        Tested app is “Benign“;
    end
 end
```

Algorithm 1: Anti-Hijack: Intent based novel algorithm

```
Data: Tested App, Benign App
Result: Malicious or benign app
initialization: Android Virtual Device, Monkey;
while App is executed with emulated or real environment do
    Execute tested app and Benign app;
    Start monkey runner tool;
end
while Read SysCall do
    Monitor benign and Analysing tested app;
    Intent or session attack detected;
    if (Permission or BroadCR S ysCall)then
        Monitor benign and Analysing tested app;
        Intent or session attack detected;
        Tested app is “malign“;
    else
        Tested app is “Benign“;
    end
end
```

Algorithm 2: Anti-Hijack: Session based novel algorithm

### 4.4. Honey Framed Benign App Design

We have designed a honey framed benign app for malign app interception and monitoring, analysing the behaviour of malicious or malign app. In order to do this, we have developed a benign app which contains user input, network ping operation, network service operation, traffic analysis, set theme concept and also include legitimate receiver in it. We have included user input in benign app for monitoring any type of user sensitive information flow leakage (Intent) and network related code for network based behaviour (Session). We have also included theme concept for monitoring non profitable behaviour of app and legitimate receiver for monitoring runtime intent based update attack.

## 5. Dataset

We have collected a large set of intent based family apps and session hijacking based apps from GNOME[7], Chinese app store [8], Virus total detected apps [12] as in Table 3, Table 4.

| Intent Based Family | # Of Apps |
|---|---|
| FakeBank | 40 |
| VMWol | 30 |
| FakeJobO er | 30 |
| CarBerp | 25 |
| Cawitt | 60 |
| Cosha | 50 |
| CruseWind | 40 |

Table 3: Intent hijack: Android malware data set

| Session Based Family | # Of Apps |
|---|---|
| DroidSleep | 50 |
| FaceNi | 60 |
| FakeAV | 50 |
| FakeFlash | 80 |
| FakeMarket | 70 |
| Obad | 60 |
| DroidDeluxe | 60 |

Table 4: Session hijack: Android malware data set

We have also collected unknown apps and some benign apps from Google app store[9], Baidu app store [10], Amazon app store [6] and Samsung app store [11] to check the efficiency of our modules as in Table 5.

| Apps Collection | # Of Apps |
|---|---|
| Intent based apps | 275 |
| Session hijack based apps | 430 |
| Unknown | 9295 |
| Benign | 10000 |

Table 5: Combined data set including benign and unknown

## 6. Results and Discussion

### 6.1. Anti-Hijack: Detection Statistics

We have tested all the data set with our proposed honey-pot device. The intent based detection is given in Table 6 and session hijack based detection is given in Table 7. We find there is a lot of impact of honey based architecture in results. Below Table 8 presents the accuracy of our model and also includes the results of benign apps as well as unknown apps (M-malign, B-benign).

| Intent Based Family | # Of Apps | Anti-Hijack Detection |
|---|---|---|
| FakeBank | 40 | 38 |
| VMWol | 30 | 27 |
| FakeJobO er | 30 | 28 |
| CarBerp | 25 | 21 |
| Cawitt | 60 | 57 |
| Cosha | 50 | 47 |
| CruseWind | 40 | 36 |

Table 6: Intent hijack: Data set results

| Session Based Family | # Of Apps | Anti-Hijack Detection |
|---|---|---|
| DroidSleep | 50 | 46 |
| FaceNi | 60 | 54 |
| FakeAV | 50 | 48 |
| FakeFlash | 80 | 77 |
| FakeMarket | 70 | 66 |
| Obad | 60 | 56 |
| DroidDeluxe | 60 | 57 |

Table 7: Session hijack: Data set results

| Apps Collection | # Of Apps | Anti-Hijack Detection |
|---|---|---|
| Intent based apps | 275 | 254 |
| Session hijack based apps | 430 | 404 |
| Unknown | 9295 | 8831 (M-5332,B-3499) |
| Benign | 10000 | 9677 |

Table 8: Combined data set including benign and unknown app results

### 6.2. Anti-Hijack: Ten-Fold Cross Validation

We have measured our approach efficiency on the basis of tenfold cross validation using precision, recall and f-measure (H-mean). We have calculated metrics on the basis of tenfold cross validation as in Table 9. Below Fig. 4 presents the accuracy and ability of our proposed frame work Anti-Hijack.

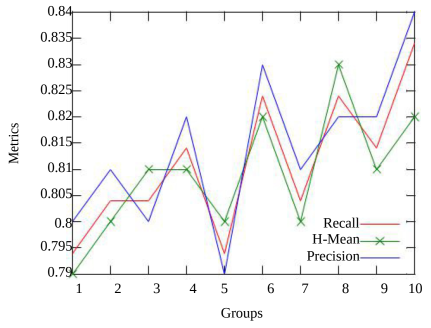


Fig. 4: Anti-Hijack: Precision, Recall and H-Mean

| Group | Recall | Precision | H-Mean |
|---|---|---|---|
| G1 | 0.79 | 0.80 | 0.794 |
| G2 | 0.80 | 0.81 | 0.804 |
| G3 | 0.81 | 0.80 | 0.804 |
| G4 | 0.81 | 0.82 | 0.814 |
| G5 | 0.80 | 0.79 | 0.794 |
| G6 | 0.82 | 0.83 | 0.824 |
| G7 | 0.80 | 0.81 | 0.804 |
| G8 | 0.83 | 0.82 | 0.824 |
| G9 | 0.81 | 0.82 | 0.814 |
| G10 | 0.82 | 0.84 | 0.834 |

Table 9: Anti-Hijack: Accuracy and ability (10-fold)

Ques:1 Why we used honey-pot based technology?
Honey-pot based technology assist low enforcement to track malign app and shut down bot-nets. It perform parallely on Android device or emulator using the concept of honey-net. This is very efficient for collecting malicious app information and detection.

Ques:2 What are the advantages of Anti-Hijack?
Anti-hijack produces less false positive because no legitimate or benign app uses honey-pot log file and there is a no need of known app’s signature so this is free from training phase.

## 7. Conclusion

Proposed approach is a strong assessment framework for exploitation vulnerabilities using run-time analysis. Primary aim of honey-pot based model is to allow malicious apps to do its activities freely. This information is used for protection of benign apps and detection of malicious apps. In this paper, we made two key contributions, first, detecting intent hijacking malware and, second, detecting session hijacking malware using our framework Anti-Hijack. In future we would like to implement some more approaches to detect root exploits and scripts which violate sandbox to gain special privileges. In terms of exploiting vulnerabilities, we used all the known vulnerabilities whose status is complete in latest SDK version of Android 4.4. Therefore, we would like to test more vulnerabilities which are not known with the latest version of Android SDK 4.4.

## 8. Acknowledgements

It is my privilege to express my sincere thanks to William Enck from NC State University, USA and Kevin Allix from University of Luxembourg, Luxembourg for supporting us during the implementation of this work. We wish to extend our gratitude to Amar Bhosale from Carnegie Mellon University for his support and consideration of our research work for appraisal.

## References

1. ADB-DUMPSYS, http://source.android.com/devices/tech/input/dumpsys.html, (Online; Last Accessed September 1, 2015).
2. Android Developers - ADB, http://developer.android.com/tools/help/adb.html, (Online; Last Accessed September 6, 2015).
3. Android Developers - Intents, http://developer.android.com/reference/android/content/Intent.html, (Online; Last Accessed August 12, 2015).
4. Android Developers - Manifest, http://developer.android.com/guide/topics/manifest/manifest.html, (Online; Last Accessed July 16, 2015).
5. Android Developers - SDK TOOLS, http://developer.android.com/tools/sdk/tools-notes.html, (Online; Last Accessed December 26, 2014).
6. Amazon Inc, Amazon Appstore for Android, http://amazon.com/mobile-apps/b?ie=UTF8node=2350149011, (Online; Accessed June 27, 2015).
7. GNOME, Malware Data Set, http://malgenomeproject.org/mobile-apps, (Online; Accessed June 25, 2015).
8. Chinese Apps, Android Malware, http://appinchina.co/the-market, (Online; Accessed May 12, 2015).
9. Google App-Store, Android Application, https://play.google.com/store?hl=en, (Online; Accessed June 02, 2015).
10. Baidu App-Store, Android Apps, https://shouji.baidu.com, (Online; Accessed May 07, 2015).
11. Samsung App-Store, Android Apps, https://findmymobile.samsung.com, (Online; Accessed May 03, 2015).
12. VirusTotal, Android Malware, https://virustotal.org/total/malware.html,(Online; Accessed April 23, 2015).
13. BakSmali, UI/Application Exerciser Monkey, http://developer.android.com/tools/help/logcat.html, (Online; Accessed April, 2015).
14. Android Overview, Open Handset Alliance - Android Overview, http://www.openhandsetalliance.com/androidoverview:html; 2015:
15. OWASP, Projects/owasp mobile security project - top ten mobile risks, https://www.owasp.org/images/9/94/MobileTopTen.pdf, 2014.
16. Wikipedia Mobile OS Marketshare, http://en.wikipedia.org/wiki/File:WorldWideS martphoneS alesS hare:png; 2015:
17. Wikipedia Mobile OS Sales, http://en.wikipedia.org/wiki/File:WorldWideS martphoneS ales:png; 2015:
18. Gartner, Gartner's annual smart-phone sales in 2014, http://www.gartner.com/newsroom/id/2665715, 2014.
19. T. B`Ieasing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, An android application sandbox system for suspicious software detection, In 5th International Conference on Malicious and Unwanted Software (MALWARE 2010), pages 55–62, Nancy, France, October 2010. IEEE Conference Publications, URL http://eprints.qut.edu.au/58112/.
20. O. C.C, Oladej, F.A.Benjamin, B.C.Alakiri, and H.A.Olisa, Penetration testing for android smartphones, October 2013, URL http://www.iosrjournals.org/iosr-jce/papers/Vol14-issue3/P0143104109.pdf?id=7444.
21. F. D. Cerbo, A. Girardello, F. Michahelles, and S. Voronkova, Detection of malicious applications on android os, In H. Sako, K. Franke, and S. Saitoh, editors, ICWF, volume 6540 of Lecture Notes in Computer Science, pages 138–149. Springer, 2010, ISBN 978-3-642-19375-0, URL http://dblp.uni-trier.de/db/conf/iwcf/iwcf2010.htmlCerboGMV10.
22. M. Chandramohan and H. B. K. Tan, Detection of mobile malware in the wild, http://www.infoq.com/articles/detection-of-mobile-malware, 2014.
23. E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, Analyzing inter-application communication in android, In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, MobiSys '11, pages 239–252, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0643-0, 10.1145/1999995.2000018, URL http://doi.acm.org/10.1145/1999995.2000018.
24. A. Cozzette, Intent spoofing on android, http://blog.palominolabs.com/2013/05/13/android-security/, 2013.
25. W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, 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, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association, URL http://dl.acm.org/citation.cfm?id=1924943.1924971.
26. R. Hay, A new vulnerability in the android framework: Fragment injection, URL https://securityintelligence.com, (Online; Last Accessed January 21,2015).
27. D. W. Kelly Casteel, Owen Derby, Exploiting common intent vulnerabilities in android applications, In MIT CSAIL Computer Systems Security Group, December 2012, URL http://css.csail.mit.edu/6.858/2012/projects/ocderby-dennisw-kcasteel.pdf.
28. J. Lessard and G. C. Kessler, Android forensics: Simplifying cell phone examinations, September 2010.
29. S. Mansfield-Devine, Android architecture: attacking the weak points, In Network Security, 2012(10):5–12, 2012, URL http://dblp.unitrier.de/db/journals/ns/ns2012.html.