ANALYSING DATA SECURITY REQUIREMENTS OF ANDROID MOBILE BANKING APPLICATION

Co-authored by

Student: Shikhar Bhatnagar

Primary advisor: Yasir Malik

Secondary advisor: Sergey Butakov

Project report

Submitted to the Faculty of Graduate Studies,
Concordia University of Edmonton

In Partial Fulfillment of the
Requirements for the Final
Research Project for the Degree

MASTER OF INFORMATION SYSTEMS SECURITY

MANAGEMENT

Concordia University of Edmonton

FACULTY OF GRADUATE STUDIES

Edmonton, Alberta

April 2018

Analysing Data Security Requirements of Android Mobile Banking Application

Shikhar Bhatnagar, Yasir Malik, Sergey Butakov
Department of Information Systems Security Management
Concordia University of Edmonton
sbhatna1@student.concordia.ab.ca

Abstract **- Mobile banking applications are at high risk of cyber attacks due to security vulnerabilities in their underlying operating systems. Android is the most popular operating system with feature like openness and customization. The Inter-Process Communication mechanism in Android enables applications to communicate, share data and reuse functionality between them. However, if used incorrectly, it can become attack surface, which allows malicious applications to exploit devices and compromise sensitive financial information. In this research, fuzzing approach is studied to analyse the data security requirement of Android mobile banking application during the inter process communication. Firstly, experimental setup automatically constructs application behaviour, after that generative fuzzing is applied to the information collected during behaviour analysis to analyse the data leak vulnerabilities. Experimental analysis and results shows the easily exploitable entry points in the applications under test.**

Keywords- Intent, Fuzzing, Generative fuzzing, Mutation fuzzing, Security testing, Data Leaks

I. Introduction

Internet and Smartphone usage continue to rise rapidly, and variety of applications and services are available for user's convenience and entertainment. Among other services, use of mobile banking is becoming common and Canadian banks are offering services and applications that allow its customer to carry out everyday banking ubiquitously through their mobile devices.

While these services and applications greatly improve productivity, they also introduce several new risks of cyber attacks. Mobile banking applications are in peril of cyber attacks due to security vulnerabilities and loopholes in the underlying operating systems. Among all, Android operating system has become a prime target for attackers as most of the Smartphone market is currently dominated by Android users. Due to the openness and customization feature in Android, applications can share resources, data and reuse functionality between them. Inter-component communication (ICC) model is partly contributing in success of collaborative framework for Android operating system [1]. However, if used incorrectly, it can become attack surface, which allows malicious applications to exploit devices and compromise confidentiality, integrity and availability of user's personal and financial information. Android operating system which is referred as Android for the rest of the paper.

The collaborative model for Android framework provides functionality like code reusability and service sharing [2]. For instance, if banking application needs to take a picture of the cheque, camera application programming interface (API) can be called to carry out this task rather than writing the whole code again. Messaging object called intent is used to make communication in between components of same application as well as different applications. The problem arises when the developer does not test the application against the security requirements/guideline provided by OWASP Mobile Application Security Verification Standard (MASVS) [3]. Attackers use evolving techniques to bypass the security mechanisms put in place by the Android to prevent user data abuse. Bankbot Trojan [4] is one such attack, which intercepts the inter-component communication to perform activity hijacking on the target financial application, and it subsequently tricks the user with fake user interface to steal the credentials [5].

The objective of this research is to analyse the data security requirements via fuzz testing the financial application's intents for data leaks. According to Google security report, the data leakage has increased up to 10 times as compared to the previous year [1], [6]. Consider a scenario in which financial application uses camera API to take picture of check to deposit money into the user account. Since no intent filter can be defined to use such service, thus malicious component such as Bankbot Trojan can attempt to hijack the activity of camera API causing potentially data loss and fraudulent transactions. This project analyses the security vetting of intents against the security requirements in OWASP Mobile Application Security Vetting Standard.

The project also intends to test the activities and services that financial applications use to transfer money via fuzzed intent injection, also verifying if this leads to data loss and determine whether it comply with OWASP mobile security best practices or not [7]. The research outcomes will help vendors, security professional and application developer to strengthen application security requirements and enhance security of their products against attacks data leak.

II. Related Works

Recently studies has been carried out to test the mobile banking applications against the exploited vulnerabilities [8], [9], [10]. In current banking application model, banks have full control over the server end, thus they have adequate measures in place to keep the data intact, but have no or little control at the user end. This make user end prone to attack and hence the scope of the research is restricted to end user security analysis.

OWASP MASVS [3] is a framework of security requirements to design, develop and test mobile applications, comprising four security verification levels namely L1, L2, L3 and L4. Where L4 being the topmost verification level and L1 being the standard security which includes the list of best security practices that any mobile application should adheres to and comply with. Level L2 consists of basic requirements and additional security controls, which are necessary for securing applications that handles sensitive information like mobile banking. The purpose of this guide is to make developers aware of security mistakes during the development phase, and focus to integrate the security in the development lifecycle. This helps to make the code more robust and end product to have less vulnerabilities. This guide can also be used as a reference for security testing methodologies.

Explored areas of testing along with the type of testing to verify the security of Android applications [3], [8], [11], [12]. Some issues/gaps were found unaddressed such as dynamic code loading, library spoofing, data capability leaks [13] etcetera. The code reusability is an Android feature that allows application developers to inherit the functionality from the code, instead of writing its own. Although, it also inherits the vulnerabilities of the third party developer's library, if imported carelessly (without testing the code). Android itself doesn't have a mechanism to detect whether used library is spoofed or benign [12]. Also, Android does not verify the loaded code integrity, which means it doesn't have the ability to differentiate if the loaded code was fetched remotely or from package itself, however the usage of latter one involves code injection attacks from other applications [14]. This makes an application exploitable, if it does not undergo regressive testing.

Study [15] points to world writable files risk. Another study [11] shows that 33 percent of the financial applications that NowSecure tested permits other applications to make changes to its files. This makes financial applications vulnerable to data leakage. In study [9] author tried to test security features of mobile banking application by exploiting man-in-the-middle vulnerability. They tested 19 Indian public sector mobile banking applications on Android platform, and observed that 90 percent of the applications failed to verify origin of certificates, and even accepted third party certificates. This led to intrusion of third party entity to eavesdrop the communication between application and server. In [16] author studied security of Android banking applications and banking server through reverse engineering the application. The temper able source code of an application which handles sensitive data is dangerous. This information can be used to apprehend the inner functioning of an application, which if misused can cause issues like SYN flood attack on bank’s server. There is also a possibility to repackage the banking application with malicious URLs to obtain user credentials by circulating it as third party application. As per to a report [17], some Canadian banks lack proper measures to detect duplication of cheque deposits. This can motivate attacker to target financial applications for fraud.

Intents are messaging objects which are used to communicate between components of applications and to trigger the components to perform some specified action. Components of Android are activity, service, broadcast receiver, and content provider (does not use intents for communication) [18]. The attributes of intent structure includes action, data, type, component and extras [19].

Financial applications that are registered to receive broadcast intent such as Alarm, Boot Completed notification etcetera without imposing control through permission. Here broadcast intent can be spoofed to cause security risk such as data leakage, as the application would not be able to identify if it came from system event or a malicious application [15].

Testing intents is important, since intents used in application component's communication are not sanitised, and can be misused by serving as attack surface, abusing the exported components of an application to carry out attacks such as intent interception, intent spoofing, intent hijacking, and data leaks. Number of testing techniques studied to perform application security vetting of intents include Taint analysis [20], [21], [22], Fuzzing [1], [13], [23], [24], [25] and reverse engineering [3].

To the best of our knowledge, fuzzers can identify the exported services and activities, which undergo stress test yielding crash reports. This is basically caused due to programming faults, where developer unintentionally fails to implement proper exception handling. In case of data leaks, studies [1], [13] closely relates to our approach. They implemented permission checking module methodology in detecting ICC data leak vulnerability. List of ICC vulnerabilities includes null pointer exception, intent interception, intent spoofing, intent hijacking, and data leaks. While this study basically covers activity hijacking, exception handling along with data leaks on the basis of scenarios discussed earlier, caused due to usage of non vetted intents communicating with exported and non-exported components in financial applications. This study aims to analyze the data security requirements of Android mobile banking applications by employing hybrid fuzz testing approach, targeting to test the Level 2 security requirements of MASVS OWASP guideline. Control flow graph was used to construct the application behavior to analyze the data flow during inter-process communication. Experiments were conducted on publically available applications for testing purpose which were side loaded on emulator. Results may vary as per to final version to application which is available to the end user. Testing financial applications Testapp4 and Testapp5 detected that the device was rooted terminating the application right away.

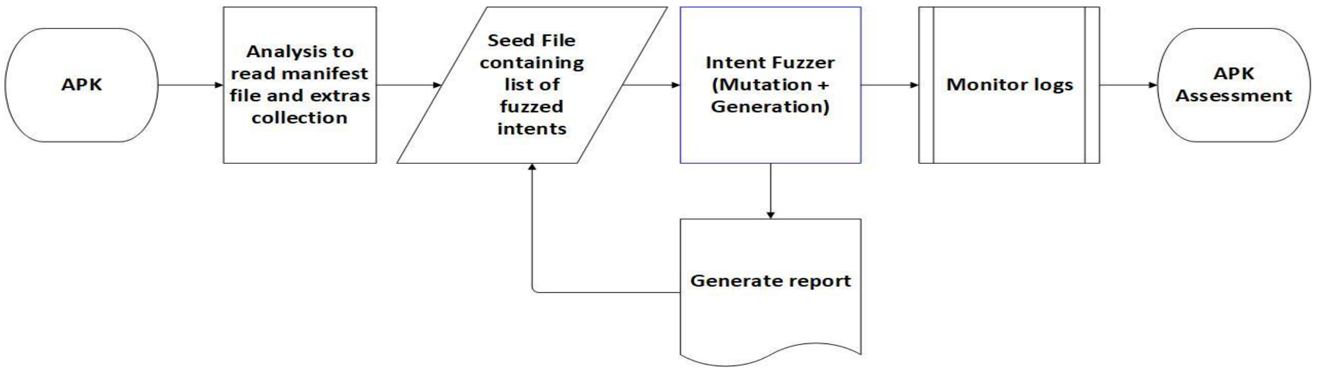
APK
Analysis to read manifest file and extras collection
Seed File containing list of fuzzed intents
Intent Fuzzer (Mutation + Generation)
Monitor logs
APK Assessment
Generate report

Figure 1 System diagram for testing setup environment

III. Experimental Setup And Used Methodology

The focus of this study is to analyse if the intents of financial applications are thoroughly vetted, so it does not leave any security loophole to allow malicious applications communicate through intent, and exploit activity/service which is not meant to be accessible. Fuzzing technique is used to find zero day security vulnerabilities in applications. Firstly, mutation fuzzing was applied to test the financial applications, and findings were later used to implement the hybrid fuzzing approach.

For mutation fuzzing test bed, Drozer [26], an open source tool by MWR InfoSecurity was used. This tool allows interacting with the underlying operating system and other applications to make use of Android Inter-Process Communication (IPC) mechanism. Some modifications were made to Drozer by adding a module named intent.fuzzinozer [27]. Drozer agent was installed on Genymotion Android emulator [28] to use server embedded on agent for establishing session between Android device and host machine. Fuzzinozer is a client side injection tool which injects intents for fuzz testing.

Fuzzinozer module provides utility of features such as broadcast intent, dos attack, run seed, select fuzz parameters and fuzzing intent. Out of which, fuzzing intent and run seed were used. For fuzzing intent functionality, target application's package name needs to be specified which automatically mutate the intents to target Application package (APK) activities. While with run seed feature, manually created list of fuzzed intents were fed to application components. The attributes of intent are mutated on basis of algorithms defined in the fuzzinozer module. The input generation method is defined for different attributes. For example, input generator function for URI is illustrated in the Snippet 1.

```
def generate_random_uri():

return random.choice(["http", "https"]) + "://"
+ str(string_generator(random.randint(1, 100)))
+ random.choice(domains)
```

Snippet 1 URI input generator function in fuzzinozer module [27]

Where the selection in between http or https is made, followed by special characters "://". And random string of value between 1 and 100 is appended, followed by domain. In the same manner, other attributes of intent are also fuzzed as per to respective pre-defined generation algorithm in the module. The fuzzinozer was used to test the financial applications and yield results like like null pointer exception, illegal argument exception and runtime exception.

This possibility to reuse this collected information as input motivated this study to fuzz test other components as well. As mentioned in Experimental setup Figure 1, this study made use of different frameworks including MARA [29], Drozer and Fuzzinozer. For hybrid (mutation + generation) testing, the methodology includes following steps:

1.Identification of exported as well as non-exported components by statically analysing the Android manifest file of the application.

2.Gather other relevant information about intent that is fuzzable such as action and type of data that it tends to receive through IPC by checking intent filter. For instance, action attribute can be fuzzed with options such as Action_Main, Action_Edit, Action_View etcetera, and to be fed as input along with other attributes targeting the component.

3.Implemented use of analyser to get the control flow graph to construct the application behaviour to analyze the data flow during inter-process communication. Also statically collected the string values for extras field in intent structure with analyser.

4.Created null intent to test the target component identified in first step and apply following fuzzer steps to find the vulnerable entry point in an application.

5.Mutation test results helped to create seed file for generative testing along with manually established number of random combinations filling up the intent attributes i.e. "**Action**", "**Data**", "**Category**" and "**Extras**" to test data flow components

6.Performed intent injection using seed file to hit the target component.

7.Monitor the logs to determine which test case caused the data leak.

8.Verify the results with Android Debug Bridge (adb) command and list down the vulnerable application.

IV. Results

Mostly fuzzinozer results represents the improper exception handling by developers. While in addition to that, experimental setup was also able to point out the architectural security weakness.

```
$  adb shell am start -n
   net.one97.Testapp2/.wallet.activity.MoneyTransfer
   Activity
```

```
Starting: Intent { act=android.intent.action.SENDTO
dat=63578888
cmp=net.one97.Testapp2/.wallet.activity.MoneyTransfe
rActivity }
```

Snippet 2 Activity Manager sending null intent to test money transfer activity in financial application Testapp2

From Snippet 3, it is evident that Testapp2 leaked Intent Receiver at "`MoneyTransferActivity`", allowing unauthorized access bypassing authentication method, causing malicious application to inject intents to target this activity and access components without any required permission. Although this activity is not even categorised as exported, still no restriction have been applied to it. The fault lies with the developer of the application as if component is marked non-exported, it should at least be protected with some permissions. Currently the exploit that this study can concur is that malicious application can use this activity to transfer money into different accounts without user's consent. All that it needs is access to add extras, money amount and phone number to transfer sum with logged in user. This possible data leaks can cause financial fraud.

```
 04-12 18:31:17.375   583   604 I ActivityManager:
 Displayed
 net.one97.Testapp2/.wallet.activity.MoneyTransferA
 ctivity: +1s150ms

04-12 18:31:17.821  3656  3676 W System.err:
net.one97.Testapp2.common.b.c:

 {"type":null,"requestGuid":null,"orderId":null,"st
 atus":null,"statusCode":"403","statusMessage":"Una
 uthorized Access","response":null,"metadata":null}
 04-12 18:32:21.476  3656  3656 E ActivityThread:
 Activity
 net.one97.Testapp2.wallet.activity.MoneyTransferAc
 tivity has leaked IntentReceiver
net.one97.Testapp2.wallet.f.c$a@9492992 that was
originally registered here. Are you missing a call
to unregisterReceiver()?

 04-12 18:32:21.476  3656  3656 E ActivityThread:
 android.app.IntentReceiverLeaked: Activity
 net.one97.Testapp2.wallet.activity.MoneyTransferAc
 tivity has leaked IntentReceiver
 net.one97.Testapp2.wallet.f.c$a@9492992
```

Snippet 3 Leaked intent receiver causing activity bypassing authentication method

And when tried to open story camera activity of Testapp2 with Snippet 4 command, application crashed.

```
$  adb shell am start -n
   net.one97.Testapp2/.social.activity.AJRUserS
   toryCamera
```

Snippet 4 Access camera activity in financial application Testapp2 caused the application to crash in Genymotion

The mutated intents injected through Fuzzinozer were able to detect exception handling errors including null pointer exception, illegal argument exception and runtime exception violating the V7: Code Quality and Build Setting requirements in OWASP MASVS guideline, leading abnormal termination of test applications.

Null pointer exception was identified in financial application Testapp2 which occurred due to null object referenced to interface method as mentioned in log file of Snippet 5.

Financial application Testapp1 crashed with intent injection on "`DeveloperConfigRcsFlagsActivity`" with the runtime exception.

Financial application Testapp3 crashed with illegal argument exception during fuzz intent injection on "`SMFeedbackActivity`" as shown in Snippet 7

```
12-04 17:14:17.100  5732  5732 F fuzzing_intent:
type: fuzzing package: net.one97.Testapp2 component:
net.one97.Testapp2.movies.activity.AJRCinemasSearchL
anding data_uri:
http://cXxw2GJyngEL5Xad4EBuMzt5j6rrs5wKW9psqmbXWn4QE
fElf94SrQKhUbsfSMC5b3xX03.gov category:
android.intent.category.CAR_MODE action:
android.intent.action.PACKAGE_RESTARTED flag:
ACTIVITY_NO_USER_ACTION extra_type: boolean
extra_key: android.intent.extra.UID extra_value:
True

--------- beginning of crash

12-04 17:14:18.223  1498  1498 E AndroidRuntime:
Process: net.one97.Testapp2, PID: 1498

--------- beginning of main

12-04 17:14:18.223  1498  1498 E AndroidRuntime:
java.lang.RuntimeException: Unable to start activity
ComponentInfo{net.one97.Testapp2/net.one97.Testapp2.
movies.activity.AJRCinemasSearchLanding}:

 java.lang.NullPointerException: Attempt to invoke
 interface method 'int java.util.Map.size()' on a
 null object reference
```

Snippet 5 Null pointer exception in financial application Testapp2

```
12-04 18:12:31.183 17198 17198 F fuzzing_intent:
type: fuzzing package:
com.Testapp1.android.p2pmobile component:
com.Testapp1.android.foundation.presentation.activit
y.DeveloperConfigRcsFlagsActivity data_uri:
http://nkPZ.com category:
android.intent.category.APP_MARKET action:
android.intent.action.TIME_TICK flag:
ACTIVITY_LAUNCHED_FROM_HISTORY extra_type: boolean
extra_key: android.intent.extra.KEY_EVENT
extra_value: True

12-04 18:12:32.303 11592 11592 E AndroidRuntime:

  java.lang.RuntimeException: Unable to start
  activity
  ComponentInfo{com.Testapp1.android.p2pmobile/com.T
  estapp1.android.foundation.presentation.activity.D
  eveloperConfigRcsFlagsActivity}:com.Testapp1.andro
  id.foundation.core.DesignByContract$DbCEnsureExcep
  tion: ### FAILED ENSURE: !!! Usage of this
  Activity is only allowed in debug mode !!!
```

Snippet 6 Runtime exception in financial application Testapp1

```
03-11 22:16:12.637 14431 14431 F fuzzing_intent:
type: fuzzing package: com.Testapp3 component:
com.surveymonkey.surveymonkeyandroidsdk.SMFeedbackAc
tivity data_uri:
http://a5mOkhTgSGvkSrHQXN43DmmLXKL3wpVFhPbqQ.mil
scategory: android.intent.category.PREFERENCE
action: android.intent.action.UID_REMOVED flag:
ACTIVITY_RESET_TASK_IF_NEEDED extra_type: boolean
extra_key: android.intent.extra.ORIGINATING_URI
extra_value: False

--------- beginning of crash

03-11 22:16:15.207 11503 11503 E AndroidRuntime:
FATAL EXCEPTION: main

03-11 22:16:15.207 11503 11503 E AndroidRuntime:
Process: com.Testapp3, PID: 11503

03-11 22:16:15.207 11503 11503 E AndroidRuntime:
java.lang.RuntimeException: Unable to destroy
activity
  {com.Testapp3/com.surveymonkey.surveymonkeyandroid
  sdk.SMFeedbackActivity}:
  java.lang.IllegalArgumentException: Receiver not
  registered: null
```

Snippet 7 illegal argument exception in Testapp3

V. Conclusion And Recommendations

This study contributes to the data security analysis using Fuzzing approach to test components that are meant to be private, but developer didn't assigned the proper permission on intent receiver. And thus forged intents can cause data security leakage as found in Testapp2. This project propose authentication and session management requirements of MASVS to validate activities and services that are not accessible through intents or any other means of communication without the authentication validation in OWASP.

The taint analysis of intents can be an effective way in detecting unintended data leakage problem in financial applications during the lifecycle of component. This approach has been matured to a great extent by peer researchers. Application developers can implement usage of this testing method in detecting application colluding through inter-app data flows which is conceivable via malware [30].

The analysis of financial applications conclude that the financial applications intents that carry sensitive information to communicate across application must be protected by permissions. Since some Trojan (Bankbot) variants [31] can detect the device it is running on and did not worked with Android emulator, and some probably did not worked with the dataset of financial applications used in this project. Study also depicts that few financial applications can detect the device it is running on, and if the running device has Google play services or not. Hypothetically, there is also a slight possibility for banking server to detect the version of application attempting to connect with. We recommend the financial applications should make user to update it to the latest version through Google play. This step can verify enforced certificate pinning and reduce the attacks carried out via application side loading.

References

[1] W. Tianjun and Y. Yuexiang, "Crafting Intents to Detect ICC Vulnerabilities of Android Apps," in *International Conference on Computational Intelligence and Security*, 2016.

[2] A. K. Jha and W. J. Lee, "An empirical study of collaborative model and its security risk in Android," *Journal of Systems and Software*, 2017.

[3] B. Mueller, S. Schleier, S. Corbiaux, J. Willemsen, A. Shrivastava, A. Temmar, A. Antukh, R. Martelloni, S. Seys, P. Singh, F. Stillavato and A. Sejpal, "About the Standard," *FOREWORD BY BERNHARD MUELLER, OWASP MOBILE PROJECT*, p. 7, 2017.

[4] Dr.WEB, "Android BankBot," [Online]. Available: https://vms.drweb.com/virus/?i=14895561&lng=en. [Accessed November 2017].

[5] Z. WANG, C. LI, Y. GUAN, Y. XUE and Y. DONG, "ActivityHijacker: Hijacking the Android Activity Component for Sensitive Data," in *25th International Conference on Computer Communication and Networks*, 2016.

[6] Google, "Android Security 2016 Year In Review," March 2017. [Online]. Available: https://source.android.com/security/reports/Google_Android_Security_2016_Report_Final.pdf. [Accessed September 2017].

[7] OWASP, "OWASP Mobile Security Project," [Online]. Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Secure_M-Development. [Accessed Novenmber 2017].

[8] S. Bojjagani and V. N. Sastry, "STAMBA: Security Testing for Android Mobile Banking Apps," in *Advances in Signal Processing and Intelligent Recognition Systems*, vol. 425, Springer, Cham, 2016, pp. 671-683.

[9] S. Kaka, V. N. Sastry and R. R. Maiti, "On the MitM vulnerability in mobile banking applications for android devices," in *IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2016.

[10] B. Panja, D. Fattaleh, M. Mercado, A. Robinson and P. Meharia, "Cybersecurity in banking and financial sector: Security analysis of a mobile banking application," in *International Conference on Collaboration Technologies and Systems (CTS)*, San Diego, 2013.

[11] C. Thompson, R. Bhatt and R. Leininger, "Mobile Banking Applications Security Challenges For Banks," 2017. [Online]. Available: https://www.accenture.com/t20170421T060949__w__/us-en/_acnmedia/PDF-49/Accenture-Mobile-Banking-Apps-Security-Challenges-Banks.pdf. [Accessed October 2017].

[12] D. Titze and J. Schütte, "Preventing library spoofing on android.," in *Trustcom/BigDataSE/ISPA*, 2015.

[13] K. Yang, J. Zhuge, Y. Wang, L. Zhou and H. Duan, "IntentFuzzer: detecting capability leaks of android applications," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014.

[14] Z. Qu, S. Alam, Y. Chen, X. Zhou, W. Hong and R. Riley, "DYDROID: Measuring Dynamic Code Loading and Its Security Implications in Android Applications," in *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Denver, 2017.

[15] F. H. Shezan, S. F. Afroze and A. Iqbal, "Vulnerability detection in recent Android apps: An empirical study," in *International Conference on Networking, Systems and Security (NSysS)*, 2017.

[16] Y. Kouraogo, K. Zkik and G. Orhanou, "Attacks on Android banking applications," in *International Conference on Engineering & MIS (ICEMIS)*, 2016.

[17] R. Marchitelli, "When banks cash your same cheque twice, you may be on the hook to pay," April 2018. [Online]. Available: http://www.cbc.ca/news/business/duplicate-deposits-mobile-chequing-banks-1.4584304.

[18] Android Developers, "Application Components," [Online]. Available: https://developer.android.com/guide/components/fundamentals.html. [Accessed 2017].

[19] Android Developers, "Intent Structure," [Online]. Available: https://developer.android.com/reference/android/content/Intent.html. [Accessed 2017].

[20] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. L. Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau and P. McDaniel, "Iccta: Detecting inter-component privacy leaks in android apps," in *Proceedings of the 37th International Conference on Software Engineering*, 2015.

[21] F. Wei, S. Roy and X. Ou, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[22] W. Klieber, L. Flynn, A. Bhosale, L. Jia and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, 2014.

[23] R. Sasnauskas and J. Regehr, "Intent fuzzer: crafting intents of death," in *Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics (PERTEA)*, 2014.

[24] Y. Wang, J. Zhuge, D. Sun, W. Liu and F. Li, "ActivityFuzzer: Detecting the Security Vulnerabilities of Android Activity Components".

[25] J. Wang, B. Chen, L. Wei and Y. Liu, "Skyfire: Data-driven seed generation for fuzzing," in *IEEE Symposium on Security and Privacy (SP)*, 2017.

[26] MWR InfoSecurity, "Drozer tool," [Online]. Available: https://labs.mwrinfosecurity.com/tools/drozer/.

[27] R. Ionescu and C. S. Popescu, "Drozer Module Fuzzinozer," [Online]. Available: https://github.com/mwrlabs/drozer-modules/blob/master/intents/fuzzinozer.py. [Accessed 2017].

[28] Genymotion, "Genymotion Android Emulator," [Online]. Available: https://www.genymotion.com/. [Accessed 2017].

[29] "MARA Framework," [Online]. Available: https://github.com/xtiankisutsa/MARA_Framework. [Accessed 2017].

[30] A. Bosu, F. Liu, D. (. Yao and G. Wang, "Collusive Data Leak and More: Large-scale Threat Analysis of Inter-app Communications," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017.

[31] Virus Total, "Bankbot Variants," [Online]. Available: https://www.virustotal.com. [Accessed 2018].