# Privacy and Security analysis of cryptocurrency mobile applications

1st Ashish Rajendra Sai
*Lero*
*University of Limerick*
Limerick, Ireland
ashish.sai@lero.ie

2nd Jim Buckley
*Lero*
*University of Limerick*
Limerick, Ireland
jim.buckley@ul.ie

3rd Andrew Le Gear
*Horizon Globex Ireland DAC*
*Nexus Center, University of Limerick*
Limerick, Ireland
andrew.legear@horizon-globex.ie

***Abstract*—Subsequent to the introduction of Bitcoin, the field of cryptocurrency has seen unprecedented growth. Mobile applications known as wallets often facilitate user interaction to these cryptocurrencies. With a perceived real world value these wallets are a target for attackers. Unlike mainstream financial services applications, cryptocurrency wallets are not subject to the same stringent security requirements of their regulated counterparts. In this paper, we examine the security profiles of commonly used Android cryptocurrency applications. We examine these applications for common vulnerabilities outlined by OWASP mobile top 10. We establish a baseline for our tests by evaluating commonly used banking and trading applications. We compare the results from our baseline test and establish the state of security provided by cryptocurrency wallet applications. The paper also examines the possible privacy implications of mobile applications. We report that the conventional financial services applications are only marginally better than cryptocurrency application in security provisions but they provide greater privacy.**

***Index Terms*—cryptocurrency, smartphone application, security, OWASP mobile**

## I. Introduction

Since the introduction of bitcoin in 2009, the field of cryptocurrency has grown to a total market capitalization of 209 billion USD [1]. These cryptocurrencies are based on *Blockchain* which is an append-only ledger. Applications known as wallets and exchanges allow user access to this ledger. A crucial deployment platform for cryptocurrency wallets is mobile phones because of the ubiquitous nature. Unlike banks, the cryptocurrency domain remains largely unregulated [2]. The lack of regulation combined with high demand may allow developers to publish applications without proper security and privacy implementation. This may prove of value to attackers as these applications often contain monetary assets and lack regulatory security requirements.

One conspicuous example of attackers exploiting the vulnerability in cryptocurrency mobile application is the ExoBot malware [3]. Other malware such as BankBot, Marcher and Mazar have also been configured to attack the cryptocurrency applications [4]. These malwares are designed to steal user credential and/or coins from the target application. Threats like these may pose an even more significant threat to the cryptocurrency domain as the user base increases. We explore the security and privacy provisions of commonly used cryptocurrency mobile applications. This research work assesses the following question:

*Are prominent cryptocurrency android applications vulnerable to common mobile security and privacy threats?*

To allow for an objective evaluation of the state of security and privacy provisions of cryptocurrency applications, we further examine ten widely used mobile banking and trading applications. The second, related, research question is:

*Are cryptocurrency Android applications any less secure than conventional financial services applications?*

The paper makes the following contributions:

- We evaluate security and privacy of cryptocurrency applications using static code analysis and network data analysis (Section 3). In static code analysis, we evaluate the source code for common security issues such as the usage of a broken cryptographic algorithm (Section 3A). Whereas with network code analysis, we monitor the application during execution and sniff the network traffic to observe the application interaction (Section 3A).
- We compare the security and privacy provisions of traditional banking and trading applications to that of cryptocurrency applications (Section 4A). We use the result to objectively evaluate the state of security and privacy provisions of cryptocurrency applications (Section 5).
- We explore the limitations of static code analysis and examine the validity of the static test for security evaluation (Section 3B).
- We propose a technique to examine the state of security provision for cryptocurrency mobile applications (Section 3B).

## II. Privacy and Security Issues

Mobile applications use operating system resources to provide services. A misconfiguration of the resource access mechanism may lead to the introduction of privacy threat vectors whereas a misconfiguration of application source code may lead to a security threat vector. In the following section, we first overview the ten most common security threat vectors

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie)

for mobile application followed by a description of privacy threats.

## A. *OWASP Mobile top 10*

The Open Web Application Security Project (OWASP) mobile security project provides a collection of the most common security issues in mobile applications [5]. The following are the top security threats to mobile applications as identified by OWASP.

- **M1: Improper Platform Usage** The mobile platform such as Apple iOS and Android provide a suite of host operating system services. Improper implementation of these services can lead to known security threats. All services provided by the host system have published guidelines for implementation, violation of these guidelines is the most common way of inflicting a known threat. For example, using App local Storage instead of iOS Keychain in iOS applications [6] to store sensitive data. App local storage may expose the information to other components of the application whereas the data stored in Keychain is secured from unauthorized access by the operating system.
- **M2: Insecure Data Storage** This category includes unintentional data leakage and insecure information storage. Data stored in local SQL databases and Log files may be prone to threat if the adversary gains access to the device. Storing sensitive data to external storage is considered insecure and can be exploited. Unlike the insecure storage media, the unintentional data leakage may be hard to detect. The data leakage may exist due to vulnerabilities present in frameworks, hardware or due to rooted devices. Applications without proper checks for data leakage can suffer from known exploitable vulnerabilities.
- **M3: Insecure Communication** Mobile applications often communicate with various services on the internet. This communication at times may involve the transmission of sensitive information such as user credentials. Bad actors might exploit vulnerabilities present in the communication to intercept sensitive information. This category includes threats induced by improper implementation of SSL for network communication.
- **M4: Insecure Authentication** Applications dealing with sensitive information such as personal or financial data should implement proper user authentication. This category of threat includes issues related to session management and user identification. One example of insecure authentication is the case where the application permits the device to execute a backend API service request without an access token.
- **M5: Insufficient Cryptography** Applications that leverage encryption often use cryptographic functions. There are two types of threats that can exploit insufficient cryptography. The first one is using a vulnerability in the implementation of the encryption/decryption process to gain access to sensitive information. The second threat occurs due to the usage of a broken cryptographic functions.
- **M6: Insecure Authorization** During authorization, the application should verify if the authenticated user has the permission to perform a given operation. Failure to do so may leave the application and back-end vulnerable. If an application allows the user to perform API calls without proper authorization, a rogue user may compromise the security of the application.
- **M7: Poor Code Quality** This category involves bad programming practices that may result in vulnerabilities such as buffer overflow. These code-level mistakes may result in the attacker exploiting business logic to bypass security provisions [5].
- **M8: Code Tampering** Mobile applications work in an environment that is not administrated by the developers of the application. This allows attackers to tamper with the code of the application to insert a backdoor or to change API calls to gain sensitive information. Attackers can modify the source code and re-sign the application before distributing it to end users [1].
- **M9: Reverse Engineering** This category involves the generation of an abstract view of the functionality of the application. This view may allow the attacker to determine the application logic in place including deduction of source code, libraries, and other assets in use. After determining the application logic, the attacker can find vulnerabilities to exploit in the design or flow of information.
- **M10: Extraneous Functionality** During the development process, the developers often use back doors for testing. This back door may involve bypassing two-step authentication during testing. Failure to remove such back doors can leave the application vulnerable.

## B. *Permission based privacy analysis*

The top 10 security threats listed in section (2A) examine how an adversary can exploit the misconfiguration of the application source code to gain confidential data. Another major privacy issue is caused by the misconfiguration of resource access mechanisms which may allow a malicious application to leak private information [7] [8]. We follow a permission-based privacy analysis model [9] to identify misconfigured resource access. We survey cryptocurrency android applications and report the system resources requested by the applications. We further investigate the application work-flow to determine if the requested service has been implemented in the application logic. In the following section, we describe the design of the empirical study used to evaluate cryptocurrency applications for the issues outlined in Section 2A and 2B.

## III. Empirical Study

For this study, we shortlist cryptocurrency applications from the Google Play Store (for an EU country) in three categories

[1]Distribution can only occur from unverified sources other than Google play store and Apple app store.

based on the number of downloads. This categorization allows us to compare the security and privacy provisions based on the prominence of the application. The first category (A) encompasses the cryptocurrency applications with more than 100,000 but less than 500,000 unique downloads. The second category (B) consists of applications with at least 500,000 but not more than 1,000,000 unique downloads. The third category (C) contains most prominent cryptocurrency applications on Google Play Store that have at least 1,000,000 unique downloads. The number of downloads is used as usage indicator. The application in category C on average tend to have a more complex code-base than other categories. They may be explained due to the greater user-base. We examine a total of 48 cryptocurrency applications over categories A, B and C. Table I provides the distribution of the participating applications over the categories. We also examine the ten most downloaded banking and trading applications from Google Play Store. Banking and trading applications are represented in table I by category D.

TABLE I
PARTICIPATION CATEGORIES

| **Category** | ***Number of Downloads*** | ***Number of Apps in testing*** |
|---|---|---|
| A | $>= 100,000$ & $< 500,000$ | 16 |
| B | $>= 500,000$ & $< 1,000,000$ | 16 |
| C | $>= 1,000,000$ | 16 |
| D | 10 Most Downloaded | 10 |

The empirical study is sectioned into two phases. In the first phase we examine the source code and network traffic for threats listed in Section 2A and 2B using automated tools. In phase two, we perform a manual analysis of source code on a subset of cryptocurrency and banking applications. Figure 1 illustrates the design of the empirical study.

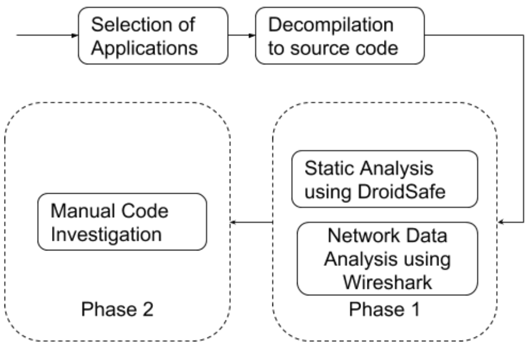
Selection of Applications
Decompilation to source code
Static Analysis using DroidSafe
Network Data Analysis using Wireshark
Phase 1
Manual Code Investigation
Phase 2

Fig. 1. Study Design

## A. Phase 1

Similar to desktop applications there are two ways of investigating the applications for security threats: static and network data analysis. In the static analysis, we examine the source code for occurrences of known security threats such as those mentioned in Section 2A. Whereas in the network data analysis of the application, we execute the application in a simulated environment and sniff the network traffic to outline any networking issue such as transmission of secret information in plain text. We now discuss the static and network data analysis approach in detail.

*1) Static Analysis:* Static analysis technique is widely used to asses the security of Android applications [10]. In static analysis, we parse the source code and understand the program path to detect program attributes. Automated tools such as DroidSafe [11] allow us to automate this process.

To perform static analysis, we require the source code of the application. We adopt the approach used by [12] to generate java source code from the Android application package file (APK). Via decompiling the packaged file, we can produce a reconstructed version of source code. This reconstructed source code can be utilized for the static analysis.

We extract the application manifest file from the decompiled source code to determine the requested system resource permissions. After identifying the requested permissions, we use the workflow of the application to determine if the permission were used during the workflow. Permissions without workflow applications have been classified as suspicious in this study. These suspicious permissions are used to evaluate the privacy implications of the application.

We now classify the threats identified by DroidSafe into the categories listed in section 2A. The automated tools such as DroidSafe can only detect a subset of the issues identified by OWASP top 10 (Section 2). Table II contains a list of all threats distinguished by DroidSafe [11] and their corresponding OWASP categories.

TABLE II
THREAT CATEGORIES

| **Threat** | ***Category*** |
|---|---|
| ELF Without Stack Protection | M1: Improper Platform Usage |
| Java Hash Code | M7: Poor Code Quality |
| Insecure Random Number Generator | M5: Insufficient Cryptography |
| IP Address Disclosure | M7: Poor Code Quality |
| MD5 | M5: Insufficient Cryptography |
| Hardcoded Sensitive Information | M7: Poor Code Quality |
| Temporary File | M2: Insecure Data Storage |
| Insecure WebView Implementation | M3: Insecure Communication |
| SHA-1 | M5: Insufficient Cryptography |
| Raw SQL Query Execution | M2: Insecure Data Storage |
| WebView Ignores SSL Certificate | M3: Insecure Communication |
| ECB mode in Encryption Algorithm | M5: Insufficient Cryptography |
| Insecure Certificate | M8: Code Tampering |

Now we review the above-listed threats in detail and report the implication of these threats on the security of the applications:

- **ELF[2] without stack protection**: This is an issue with the binary files of the application. An application built with -fstack-protector option can significantly increase the difficulty of a stack overflow [13].

[2]Executable and Linkable Format (ELF) is a standard for executable files in UNIX

- **Java HashCode**: hashcode() function in java allows users to generate a hash value of 32-bit. Due to the 32-bit size limit, Java hashcode can generate 4 billion unique hash values. By applying birthday attack [14] on java hash code, we ascertain that with 77000 objects we have a 50% chance of collision. This proneness to collision renders java hash code insecure for cryptographic hashing.
- **Insecure Random Number Generator**: Using a predictable random number generator for operations such as key generation can prove to be insecure. In Java, the Random.nextInt() is deemed insecure because of its predictable nature [15].
- **IP Address Disclosure**: Disclosing IP addresses of the backend server may provide an attacker with more attack vectors such as exploitation of the web server to gain sensitive data.
- **MD5**: MD5 hashing algorithm has been broken and is considered weak [16]. Even though there are acceptable use cases for MD5, it may still pose a security threat if it is used in critical cryptography algorithms in the application.
- **Hardcoded Sensitive Information**: Sensitive information such as API keys may reveal the structure of the backend resulting in more attack vectors.
- **Temporary File**: Storing sensitive information in temporary files can lead to data leakage if the attacker gains access to storage.
- **Insecure WebView Implementation**: Improper implementation of WebView in Android can lead to an insecure direct object reference and Cross-Site Scripting.
- **SHA-1**: [17] have proposed an attack on SHA-1 which undermines the security promises of SHA-1 insecure for usage in critical security functions.
- **Raw SQL Query Execution**: Allowing raw SQL queries may lead to a SQL injection. SQL injection is particularly harmful if sensitive data is stored in the SQL database.
- **WebView Ignores SSL Certificate**: Ignoring SSL certificate in WebView can allow an attacker to conduct a Man-in-the-middle attack [18].
- **ECB mode in Encryption Algorithm**: Electronic code block is a type of block cipher technique in which at a time a block of plaintext is translated into ciphertext. ECB generates the same ciphertext for the same substring if processed again. Repetition of the same ciphertext may allow a cryptanalyst to deduce plaintext from the ciphertext.
- **Insecure Certificate**: Using algorithms such as MD5 and SHA-1 to generate a digital certificate may not be secure due to the algorithms used.

We extend the static code analysis by detecting security features implemented by the developers. We modify the DroidSafe to detect the following security features.

- Root Detection: Rooted Android devices can bypass many security provisions, detecting a rooted device allows the developer to handle such threats.
- Prevention of tapjacking attack: In a tapjacking attack, the attacker attempts to record the touch events of other applications using a malicious application on the device. This attack may allow the attacker access to typed credentials. We detect the existence of code to prevent such an attack in the source code.
- SSL Pinning Library: SSL pinning can improve the security of a service that relies on SSL certificates. By using a pinning library, the developer can prevent a Man-in-the-middle attack.
- Database with encryption: An encrypted database is not prone to data leakage via SQL injections.

*2) Network Analysis:* To detect potential communication vulnerabilities (M3: Insecure Communication), we use a simulated application execution environment and monitor data flow from the application. We use the Android emulator [19] and route the data traffic through Wireshark [20] for packet analysis. In the network data analysis, we observe the traffic for sensitive information such as credentials or API keys.

### *B. Phase 2*

The static code analyzer (DroidSafe) examines the source code to report usage of insecure methods in code implementation. The static code analysis enables us to identify occurrences of known threats (listed in table II) in the decompiled source code. The severity of these identified threats depends on the functional use of the threat prone component, for example, if the application is using MD5 to generate the hash of a document, it does not pose a significant threat to the user. The static analyzer fails to examine the functional implication of the threat prone component.

We aim to address this threat to validity by examining a subset of the test applications manually to determine the detected threats security implication. More specifically, we examine the following two components of the test application for the presence of the aforementioned threats: Authentication and Cryptography. The nature of cryptocurrency applications requires a high dependence on cryptography functions. A faulty cryptography library or implementation may lead to a vulnerable application. The same is true for the authentication components, as a compromised authentication components can lead to a leak of sensitive user data and let attackers bypass authentication or authorization constraints.

For the manual testing, we locate the cryptography and authentication components in the application by using a hybrid feature location technique [21]. After locating the components responsible for authentication and cryptography function, we examine them for occurrences of threats outlined in Section 2. We report the findings of our manual analysis in Section 4B.

## IV. Results

In the following section, we report the results from the empirical study conducted in Phase 1 and Phase 2. In the results of the Phase 1 study, we first establish a baseline for objective comparison of the results. For baseline testing, we perform the tests described in Phase 1 on the ten most downloaded banking applications on Google Play Store (for an EU country). We also report the test results for the shortlisted cryptocurrency applications based on their categories. Following the results of Phase 1 of our study, we report the results of manual testing of four cryptocurrency and four banking applications.

## A. Phase 1

By following the permission-based privacy analysis model [9] we report that all the tested applications required access to the internet. 98% of tested cryptocurrency applications require access to the camera; this may be due to the ability of scanning QR codes to input an address. We report that all the applications requesting the camera had implemented it in the application workflow. Table III lists the most common permissions requested by the test applications.

TABLE III
THREAT RESULTS

| **Permission** | **Application Category** | | | |
|---|---|---|---|---|
| | ***A*** | ***B*** | ***C*** | **D** |
| Internet | 100% | 100% | 100% | 100% |
| Write To External Storage | 92% | 70% | 75% | 80% |
| Camera | 92% | 70% | 92% | 90% |
| Access To The Network State | 84% | 90% | 84% | 60% |
| Fingerprint | 44% | 50% | 48% | 50% |
| Request User Accounts | 32% | 30% | 34% | 30% |
| Wake Lock | 72% | 70% | 75% | 70% |
| Cloud Messages | 36% | 50% | 58% | 60% |
| NFC | 8% | 30% | 16% | 10% |
| Read SMS | 4% | 0% | 25% | 10% |
| Read Contacts | 16% | 10% | 41% | 50% |
| Bluetooh | 8% | 10% | 16% | 20% |
| Access GPS | 24% | 10% | 8% | 50% |
| **Average Result** | **47%** | **45%** | **52%** | **52%** |

The permission-based privacy analysis also revealed some suspicious permissions that lacked any work-flow implementation. Two cryptocurrency applications in our test set requested permission to record user audio which had no workflow implementation. Another equally alarming suspicious permission request was for reading call logs. One application in our test set requested permission for reading call logs. Two test applications requested permission to read system logs. One application requested permission to initiate a phone call on behalf of the user. These suspicious permissions can lead to leakage of undesirable user information without the explicit consent of the user. Unlike cryptocurrency applications, banking applications reported no suspicious permission requests.

Permission requests to write to external storage directly corresponds to M2 in OWASP mobile top 10. Other conventional security threats among test applications include the presence of Java HashCode in 82% of tested applications. We summarize the test report in Table IV.

We can observe from Table IV that applications belonging to category B tend to have a higher proneness to threats than the other three categories. Whereas applications belonging to category A tend to have lowest threat proneness, this may be due to the less complex nature of the applications falling in this category. Applications in category C tend to perform better in static code analysis than applications of category B; this may be due to their more significant user base and stricter security policies. On an average, both the cryptocurrency and banking applications report to perform similar in the tests performed.

TABLE IV
THREAT RESULTS

| **Threat** | **Application Category** | | | |
|---|---|---|---|---|
| | ***A*** | ***B*** | ***C*** | D |
| ELF Without Stack Protection | 60% | 30% | 50% | 60% |
| Java Hash Code | 84% | 80% | 94% | 90% |
| Insecure Random Number Generator | 68% | 90% | 75% | 80% |
| IP Address Disclosure | 84% | 80% | 66% | 50% |
| MD5 | 84% | 80% | 84% | 50% |
| Hardcoded Sensitive Information | 72% | 70% | 75% | 70% |
| Temporary File | 40% | 60% | 41% | 20% |
| Insecure WebView Implementation | 32% | 70% | 34% | 30% |
| SHA-1 | 48% | 70% | 66% | 20% |
| Raw SQL Query Execution | 52% | 30% | 66% | 60% |
| WebView Ignores SSL Certificate | 4% | 10% | 25% | 10% |
| ECB mode in Encryption Algorithm | 16% | 20% | 34% | 20% |
| Insecure Certificate | 12% | 30% | 25% | 20% |
| **Average Result** | **50%** | **55%** | **57%** | **45%** |

Apart from the threats we also report the security features implemented by the applications. In our cryptocurrency test applications, 58% of cryptocurrency applications were able to detect a rooted device (56% in category A, 60% in category B, 62.22% in category C and 90% in category D). Only 4% applications had capabilities to detect a tapjacking attack. Only two applications from our test sets use an encrypted database and SSL pinning libraries. This finding is alarming as these cryptocurrency applications often contain monetary assets.

It is worth noting that the severity of the threat may depend on the implementation of threat prone component. As outlined by OWASP top 10, using a temporary file as a storage media may not be secure, but it does not necessarily pose a significant threat as long as no sensitive information is stored in it. Other threats such as MD5 and SHA-1 may not pose a severe threat as long as they are not used in any security sensitive component of the application. To assess this more deeply, in a pilot fashion, in phase 2 we look more deeply at these issues for a subset of the application.

## B. Phase 2

In phase 2, we perform the manual investigation of the source code and determine if the issues detected by static analysis in phase 1 pose any real security threat to the authentication and cryptography component of the application. We examine the four most downloaded cryptocurrency applications and four most download banking and trading applications. We manifest the average results from our testing in table V.

The results of manual investigation suggest that static code analysis can be misleading as most detected issues do not pose a direct security threat to the user. We noticed that over half of the issues detected by Phase 1 analysis were not harmful to the end user. Even though the limitation of static code analysis limits the validity of results, the trend towards

TABLE V
THREAT RESULTS

| **Test Category** | **Application Type** | |
|---|---|---|
| | ***Cryptocurrancy*** | ***Banking and Trading*** |
| Issues detected by Phase 1 | 11 | 10 |
| Issues in Authentication | 2 | 2 |
| Issues in Cryptography | 2 | 1 |
| Security Features | 3 | 1 |

security seems to remain the same. The banking and trading applications in our manual investigation performed slightly better than cryptocurrency applications. The overall number of issues flagged by static analysis were lower for banking and trading applications than cryptocurrency applications. This trend is most prominent in the cryptography component of applications, where cryptocurrency applications reported a higher number of threats.

## V. CONCLUSION

In this paper, we performed an evaluation of the security and privacy of cryptocurrency applications using static code analysis and network data analysis (Section 3). Results from the static and network data analysis suggest that traditional banking applications have a lower rate of security vulnerabilities compared to cryptocurrency applications. The most common reported issue is insufficient Cryptography (M5) followed by insecure data storage. The insufficient cryptography issue is relatively easy to address by exploiting modern cryptography functions such as SHA2. Insecure data storage issue can be addressed by ensuring no temporary files contain sensitive information and also by using an encrypted database for the application. The difference between security provisions of cryptocurrency and banking and trading applications as measured by Phase 1 of the study is marginal. However, the permission-based privacy analysis reported 8% of tested cryptocurrency applications had one more malicious permissions compared to no malicious permission request by banking applications. The results of Phase 1 suggest that cryptocurrency and banking applications provide the same level of security but banking applications tend to provide higher user privacy standards than cryptocurrency applications.

We also report the limitations of static code analysis and how it may be misleading as a security evaluation tool. The application of detected threat vectors in non-security sensitive areas of applications may not pose a security threat, but the static code analysis approach does not consider this. This limits the application of static code analysis approach to evaluate the state of security for an application space.

We propose and demonstrate a technique to examine the state of security provision for cryptocurrency mobile applications (Section 4B). In this technique, we use a hybrid feature location technique to locate critical security components of the application (authentication and cryptography) followed by a manual investigation of the detected components for the presence of security threats. We report that over half of the threats detected by static analysis do not pose any direct security or privacy implication. We also report that the state of security provision for cryptocurrency and banking applications is very similar, but banking applications tend to have half as many cryptography related issue as cryptocurrency applications.

The study presented in this paper is limited by the choice and number of participating cryptocurrency and banking applications. We have divided the participation over the number of downloads to address the cryptocurrency application vertical as a whole. The limited number of banking and trading applications (currently capped at 10) may pose a threat to the reported results. We intend on conducting the study on a larger data-set in the future to eliminate this limitation.

## REFERENCES

[1] "Cryptocurrency market capitalizations — coinmarketcap," Oct 2018. [Online]. Available: https://coinmarketcap.com/

[2] A. Nelson, "Cryptocurrency regulation in 2018: Where the world stands right now," *Bitcoin Magazine. Online at https://bitcoinmagazine.com/articles/cryptocurrency-regulation-2018-where-world-stands-right-now*, 2018.

[3] P. D. Phuc, "Threatfabric." [Online]. Available: https://goo.gl/Kx7QbU

[4] L. Kessem, "Mobile banking trojans as keen on cryptocurrency as pc malware," Mar 2018. [Online]. Available: https://securityintelligence.com/mobile-banking-trojans-as-keen-on-cryptocurrency-as-pc-malware/

[5] "Owasp mobile security project." [Online]. Available: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10

[6] "Keychain services." [Online]. Available: https://developer.apple.com/documentation/security/keychain_services

[7] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[8] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *International conference on Trust and trustworthy computing*. Springer, 2011, pp. 93–107.

[9] Z. Fang, W. Han, and Y. Li, "Permission based android security: Issues and countermeasures," *computers & security*, vol. 43, pp. 205–218, 2014.

[10] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of android apps: A systematic literature review," *Information and Software Technology*, vol. 88, pp. 67–95, 2017.

[11] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard, "Information flow analysis of android applications in droidsafe." in *NDSS*, vol. 15, 2015, p. 110.

[12] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," *Proc. of Black Hat Abu Dhabi*, pp. 77–101, 2011.

[13] "[patch] add a new option "-fstack-protector-strong" (patch / doc inside)." [Online]. Available: https://gcc.gnu.org/ml/gcc-patches/2012-06/msg00974.html

[14] M. Girault, R. Cohen, and M. Campana, "A generalized birthday attack," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1988, pp. 129–156.

[15] "Insecure randomness." [Online]. Available: https://www.owasp.org/index.php/Insecure_Randomness

[16] H. Dobbertin, "The status of md5 after a recent attack," *Crypto-Bytes The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.*, vol. 2, no. 2, 1996.

[17] M. Stevens, P. Karpman, and T. Peyrin, "Freestart collision for full sha-1," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 459–483.

[18] Y. Desmedt, "Man-in-the-middle attack," in *Encyclopedia of cryptography and security*. Springer, 2011, pp. 759–759.

[19] "Android emulator — android developers." [Online]. Available: https://developer.android.com/studio/run/emulator

[20] A. Nath, *Packet Analysis with Wireshark*. Packt Publishing Ltd, 2015.

[21] W. A. E. C. Razzaq, Abdul and B. Jim, "The state of empirical evaluation in static feature location," *ACM Transaction on Software Engineering Methodology*, vol. 28, no. 1, p. TO APPEAR, 2019.