Cryptocurrency Security Study based on Static Taint Analysis

Anyu Yang *

University of Southern California, Los Angeles, America

* Corresponding author email: anyuyang@usc.edu

Abstract. Cryptocurrency represented by Bitcoin is a very popular topic in recent years. However, the prosperity of cryptocurrency drives an increasing number of applications published. Some malicious or vulnerable programs are also detected and reported these years. To do a deeper study into security of cryptocurrency application, this paper learns common vulnerabilities, threat models inside normal applications, and taint analysis, a useful vulnerability-detecting tool, concludes a common and useful methodology for threat detection in application programs, especially Android apps. This approach uses static taint analysis to detect vulnerabilities inside a given Android application, classify them into common vulnerability categories and then make conclusions. This paper does research in analyzing statistics of threats in common cryptocurrency apps in Google play store and draw conclusions on the status of cryptocurrency software as well. Finally, some suggestions are provided at the end of this paper. These recommendations apply to application programmers, app store administrators, scholars and experts in related area, government officer and users. This set of analysis process can be applied to analyze any type of application programs.

Keywords: Cryptocurrency Security; Vulnerability Categories; Threat Models; Static Taint Analysis.

1. Introduction

The prosperity of Internet and thriving of world Economy promote the development of cryptocurrency. Up to Nov 2022, more than 21,000 kinds of cryptocurrency are active in market [1]. However, the prosperity of cryptocurrency does not mean most of cryptocurrency are safe enough. According to statistics from Federal Trade Commission website [2], From the Jan 2021 to June 2022, more than 46,000 people have reported they have experienced cryptocurrency fraud, and more than billion USD has lost in these frauds. Therefore, it is worthy digging potential vulnerabilities or risks behind those cryptocurrencies. Reasons of frequent cryptocurrency problems could be divided into two categories: technical aspect and social engineering aspect. Social engineering aspect contains reasons tightly related to financial fields, such as Ponzi Schemes and giveaway frauds. These will be roughly discussed in section five and this paper will mainly focus on technical aspect.

From technical aspect, most of cryptocurrencies derive from bitcoin and block chain, which is an append-only ledger [3]. Cryptocurrency applications, which are also called wallets or exchanges, provide a platform for users to access their ledgers, where users could make transactions. Due to its widespread use, cellphone becomes an indispensable deployment platform for cryptocurrency programs. One of advantages advertised by those cryptocurrency applications is decentralization. However, decentralization also means lack of effect and efficient regulation and standard. Vulnerable programs might be published on the market. Even worse, some malicious programs camouflage themselves as cryptocurrency applications, if users download one of them, private data will leak. Marcher (aka Exobot) [4], a banking Trojan, could be configured to attack cryptocurrency applications. These malicious programs might steal users' credential and coins or utilize permission of cryptocurrency applications to get other sensitive information on cellphones. Given various problems about cryptocurrency applications and vulnerable programs on the market, it is necessary to do some research in this area. Therefore, the paper studies data on common vulnerabilities in Android apps and makes the following contributions:

1) This paper lists common vulnerability categories and threat models of applications related to Web, concludes methods to analyze and test application programs, and points out a clear direction to analyze any kind of applications in the future. Readers could use this methodology in analyzing any kind of applications.

2) This paper analyzes statistics of vulnerabilities inside Android programs and discusses some common but hidden loopholes inside cryptocurrency programs. This may give advice to cryptocurrency application programmers or Android platform administrators.

3) This paper also gives some further suggestions to cryptocurrency applications, not only from application programmers' perspective, but from regulators' perspective as well. These recommendations are not limited to cryptocurrency domain, but can be applied to any field with decentralization, such as Web 3.0 and self-media.

2. Vulnerability Categories

The Open Web Application Security Project® (OWASP), a non-profit organization in software security and Web security, lists "Top 10 Web Application Security Risks" every year, which could be used as an excellent standard. The following displays the OWASP's Top 10 ranking list in 2021[5].

1) Broken access control

Access Control is a kind of security mechanism to ensure security policies, which authorizes legitimate users to have ability to access given resources in the system and prevents unauthorized information flow. According to OWASP [5], 94.55% of applications tested by them were detected having the vulnerability of broken access control with the average incidence rate of 3.81%. They also listed some common instances including "Exposing sensitive data to unauthorized actor", "inserting sensitive data into sent packet" and "Cross-Site request forgery".

2) Cryptographic failures

Cryptographic Failures, also named Sensitive Data Exposure, acts as one kind of major vulnerabilities inside computer systems. Cryptography is used to scramble data, authorize a given user and verify integrity, so cryptographic failures will influence nearly every aspect of a system. Cryptographic failures include crypto algorithm being cracked, insufficient entropy (shorten key space) and lack of encryption.

3) Code injection

Injection-related web application vulnerabilities often has one explicitly-defined dangerous functions, which is named sink function [6], including echo for cross-site scripting (XSS) and move_uploaded_file for arbitrary file upload. Attackers could utilize these functions and insert some well-designed code segments into their input to a web server, and then the web server will execute this code segment or view this code segment as part of its instruction, which breach the original security policy of the server. Some common injections include SQL, NoSQL, OS command or LDAP injection. According to OWASP, 94.04% of applications they tested has detected loopholes for injection attack.

4) Insecure Design

Insecure design is a brand-new category in the ranking list of 2021. This broad classification represents various weakness and be expressed as "missing or ineffective control design". Insecure design, not being same as insecure Implementation, is a kind of born defect of applications or systems. Secure design might also have vulnerabilities by mistakenly implementing, but this could be patched or fixed up. However, insecure design is much harder to be fixed up or even be unable to be solved at all.

5) Security Misconfiguration

The ranking of this category has moved up these years. This is unsurprising because an increasing number of users are in love with configurable software or the default configuration of applications. As a result, many users unknowingly authorize apps in their computers or cellphones to have access to secret they are not supposed to gain, such as location, address book and preference information.

6) Vulnerable and outdated components

This category mainly contains use of discarded, out-of-dated or vulnerable components from third party, or users might install some untrusted plug-ins in their system, which leave backdoors inside their system. For example, the heart bleed bug [7], a serious vulnerability inside OpenSSL library,

was fixed after OpenSSL 1.0.1g released on 7th of April 2014, but if users installed OpenSSL 1.01 through 1.01f or some operating system distributions with potentially vulnerable OpenSSL version like Ubuntu 12.04.4 LTS or Fedora 18, they will still be attacked by Heartbleed.

7) Identification and authentication failures

Authentication is a process that combine an identity to a subject, according to what the entity knowing, what it is having and its biometrics. Identification and authentication weakness defined by OWASP mainly contains permission of weak or well-known password, reuse session identifier after successful login or no cancellation of SSO after logging out. Using single password is not same as cryptographic failures before. In cryptographic failure, even if users set some complicated keys that cannot be cracked using brute-force, vulnerabilities also exist inside the system.

8) Software and data integrity failures

This classification mainly concentrates on making assumptions about software updates, critical data and CI/CD pipelines without integrity verification. Users might download and install untrusted software without integrity check. In addition, the man in the middle attack will also exploit integrity failures [8]. The SolarWinds Orion (SUNBURST) cybersecurity breach [9], as an example, happened in Mar 2022. Malicious code updated onto the Orion Platform were pushed to almost 18,000 customer networks, who became potential victims.

9) Security logging and monitoring failures

Security logging and monitoring failures can be hard to test, but this will cause the failure of intrusion detection. As a result, Failure of monitoring and logging will not have a huge influence to a system directly, but failure of intrusion detection means the system being unable to report and defense further intrusion and damage. This category includes no logging auditable events, log file tampering and no monitoring for suspicious activity, etc.

10) Server-Side request forgery (SSRF)

Server-side request forgery [10], also known as SSRF, is a web security vulnerability, which could allow attackers induce the server-end application programs to make requests to unintended locations. A successful SSRF attack will violate access control policy and causes the leak of sensitive data on the server. The worse situation is some SSRF might execute specified instructions on the server-end or get into back-end system.

3. Threat Models and Methodology

3.1 Taint Analysis

Taint analysis is able to track sensitive data flows, also called taint flow, within software programs and widely used in detection of vulnerability, privacy leak, and malicious codes [11-14]. In real-world scenarios, private data flow might leak to public sinks, such as Internet and SMS transmission, taint analysis helps to detect this kind of leakage. Taint analysis simulates a sensitive data flow, which derives from its source, propagates through the application. When the data flow reaches to a sink point, simulator knows it. Most of works in this area could be divided into three categories: Static taint analysis, Dynamic taint analysis, and Hybrid taint analysis [15-17]. As stated in [11], dynamic taint analyses propagate taints when applications are running. This simulates the real program execution and could always find the true taint flows. Nevertheless, because of conditional judgment and jump instructions inside programs, dynamic approach might not reach all control flows of applications and will recur run-time performance overhead. On the other hand, Static taint analysis propagates taints on the static source code basis. For this reason, this approach overestimates all possible program flows and has the capacity to find out all potential taints. Nevertheless, this stands a high chance to bring in some false positives and infeasible paths. Hybrid approach combines both static and dynamic approach and using one as major method and the other one as an auxiliary mean. Since false positives is acceptable in this research and static approach is easier to be implemented, this paper mainly focuses on static approach.

In static taint analysis, automated tools such as DroidSafe [18], FlowDroid [19], and Amandroid [20] could be used to analyze taint flows. Method stated in [21] could be used to retrieve the reconstructed version of Java source code from bytecode of applications, more concretely, the Android application package file (APK), by using decompiling. Then the output source code could be used in static taint analysis.

3.2 Common Threat Models

Section two lists top 10 vulnerability categories. However, these categories only describe threats from an abstract and overall perspective. For example, the A3. Code injection contains various kinds, such as SQL injection, OS commands, XML parsers, etc. It is hard and unwise to design and create a system to detect all vulnerabilities inside a particular category. Moreover, some categories in section II are not directly related to client ends, like A10. SSRF, or they need to be considered from systems' perspectives. Therefore, it is wiser to lists common threat models inside Android applications, detecting whether they exist inside cryptocurrency applications, and finally classifying them into these general categories and make conclusions.

The following states some common threats inside Android applications, which could be detected by static taint analysis.

1) ELF without stack protection:

Executable and Linkable Format, created by the Tool Interface Standards (TIS) community [22], is a file format for object files, program loading and dynamic linking files, and shared libraries. The most common vulnerability in ELF file is the stack overflow, in which the runtime stack pointer exceeds the stack bound, which is set up by the compiler. Unintentional stack overflow might cause the crash of process while well-crafted stack overflow will lead to jump of EIP and leak of sensitive data.

2) Insecure random number generator

Random number is a critical part in key generation. In order to be safe, random number should be truly random, messy, unpredictable and non-repeatable. However, hardly can this be implemented in a deterministic algorithm. So most of case, people use pseudo random number generator. However, this part might be a loophole inside the application, especially when programmers use some simple and predictable random number generators. If adversaries gain the so-called random number, it will be not hard for them to find the key.

3) MD5 or SHA1

MD5 produces a 128-bit hash value while SHA1 produces a 160-bit hash value as outputs, but they are both proved to be unsafe [23]. To be honest, using SHA256, which is 256-bit hash value is not a wise choice in some cases. If applications use MD5 or SHA1, it could be cracked by dictionary attack [24], rainbow table attack [25] or mask attack. Password attacking tools, such as hashcat [26], supports various types of hash attacking. As a result, using weak hash functions to generate hash value inside applications could be seen as a vulnerability.

4) IP address disclosure

Exposure IP address to attackers might incur further attack. For example, adversaries could do port scan, which could help them do inversion in the future. In addition, IP address disclosure might result in loss of sensitive information [27].

5) Unencrypted sensitive data

Some applications might hardcoded sensitive data, such keys or important APIs, or store them in the form of plain text. This will increase the attack surface, the accumulation of every parts of the system which are exposed to attackers.

6) Temporary file

During run time, application processes might create some temporary files to store sensitive data. Nevertheless, seldom do application programmers encrypt these temporary files because they will delete these files after finishing. Sensitive data like user's secrets might lose confidentiality if adversaries get the priority to access file system.

7) Raw SQL query execution

When interacting with servers, users need to provide input and servers might use this input as part of SQL sentence and do some lookup operation inside the backend database system. However, if the server does not check users' inputs, some well-designed inputs could steal or even delete sensitive data inside database.

8) No use of SSL/TLS

Secure Sockets Layer (SSL), developed in 1995, is used for building an encrypted secure link between two-ends, typically between server and client. TLS builds on the now-deprecated SSL specifications and is more secure than SSL. SSL/TLS uses handshake for exchanging keys and building up secure connection. If applications does not use SSL/TLS, messages between server and client will be in the form of plain text, which stands a high chance of information leakage.

9) Insecure certificate

Certificate contains server's public key, which is used in further message transformation. In order to verify certificate, public key from a trusted third party is needed. If insecure certificated is used, attackers might add their own certificate inside users' web browsers, which could be utilized by the man in the middle attack.

10) Unsafe encryption mode

Original versions of symmetric encryption algorithm, such as DES and AES, are often block cipher, also called electronic codebook (ECB) mode. In ECB mode, inputs are divided into certain length blocks and each block is ciphered separately. However, ECB mode is provably insecure, because when 2 blocks are same, the outputs will also be same. This shortcoming is especially apparent in image encryption. More optional choice is to use stream cipher, including Cipher block chaining (CBC), Propagating cipher block chaining (PCBC), and Cipher feedback (CFB).

11) Insecure web view implementation

WebViews are used in Android applications, which loads HTML or other contents within applications. Application programs might leave some common loopholess. If contexts inside WebView are loaded in the form of plain text, attackers is able to read the data. In addition, if WebView has the world readable permission, adversaries could inject a malicious html file by using file:scheme to access local files of the user.

4. Statistics

Discussing vulnerability categories, analysis methodologies and specific threat models lay the foundation for researchs and data analysis. Table 1 records frequencies of each threat detected in each categories. Categories are divided according to the number of downloads in Google Play Store [28], A contains apps with 100,000 to 499,999 downloads, B stands for 500,000 to 999,999 downloads programms, C includes applications whose downloads are greater than 1,000,000, and D represents ten most downloaded application programs. Table 1 also shows the average probability of detecting a threat model in each category, which displays an intuitive and concise conclusion about each category, in spite of the fact that different threat models pose a different level of influence to users. Averages in Table 1 shows that number of vulnerability in each category have no strong relation to the number of downloads, which means concurrencies prefered by users do not do better in shrinking attack surface. Also, concurrency applications do well in using SLL/TLS in HTTP protocols, using secure certificates and safe encryption mode, which are not so difficult to be added into application programs due to many convenient packets and libraries on the market. However, most concurrency applications have loopholes in using insecure random number generators, IP address disclosure, having part of sensitive data unencrypted and using MD5 or SHA. This situation could be explained by two reasons. First of all, theses security mechanisms are difficult to be implemented. Secondly, those application programers might not have a strong security awareness, they might ignore parts of sensitive data produced during the runtime of programs and use packets or tools from internet without any modifications.

Table 1. Threat incidence by application category

Threaten	Application Category				
	A	B	C	D	Average
ELF without stack protection	60%	30%	50%	60%	50%
Insecure Random Number Generator	68%	90%	75%	80%	78%
IP Address Disclosure	84%	80%	66%	50%	70%
Unencrypted sensitive Data	72%	70%	75%	70%	72%
Temporary File	40%	60%	41%	20%	40%
Raw SQL Query Execution	32%	70%	34%	30%	42%
No use of SSL/TLS	4%	10%	25%	10%	12%
Insecure Certificate	12%	30%	25%	20%	22%
unsafe encryption mode	16%	20%	34%	20%	23%
Insecure WebView Implementation	32%	70%	34%	30%	42%
MD5 or SHA1	84%	80%	84%	50%	75%
Average	46%	55%	49%	40%	48%

After classifying these threatens into categories listed by OWASP, some conclusions could be made. The classification is shown in table 2.

Table 2. Classification of threatens

Threaten	Category	
ELF without stack protection	A4	A6
Insecure Random Number Generator	A2	A6
IP Address Disclosure	A4	
Unencrypted sensitive Data	A2	
Temporary File	A2	A4
Raw SQL Query Execution	A3	
No use of SSL/TLS	A7	A8
Insecure Certificate	A7	A8
Unsafe encryption mode	A2	A6
Insecure WebView Implementation	A3	A6
MD5 or SHA1	A6	A8

In fact, access control is an abstract and generic definition, all vulnerabilities in table 1 might lead to further compromises to the system and any compromise of the system is tightly related to breaking of access control. Moreover, the configuration of applications is done when users download applications, which might be avoided if users read terms and conditions carefully. And security misconfiguration is reflected in table 4 [29]. As for A9 and A10, A9 had better be done from operating system level and A10 is for server-end. As a result, A1, A5, A9 and A10 disappear in table 2 because they do not have direct relation to these threatens. However, they also need to be considered in cryptocurrency applications. In addition, most of threats could be classified into more than one category, for example, using MAD5 or SHA1 means using vulnerable and outdated components, (A6), but using this kind of vulnerable component will cause the failure of integrity, which means this could also be recognized as A8. Combining Table 1 and Table 2 could generate Table 3, which describe an overall perspective of vulnerability categories inside applications. To calculate the probability of one category being detected inside one Android application, suppose each threat are mutually independent, this is a bit inaccurate, but could make intuitive conclusions.

Table 3 shows that cryptographic failures, insecure design, vulnerable and outdated components, identification and authentication failures are common in most of applications. This is not surprising because the judging criteria are very strict in this research. For example, SHA1 and MD5 are widely used in some applications which have low security requirement. In addition, one threat model might break several policies and could be classified into more than one vulnerability categories, so these classifications have relations with each other. Apart from security vulnerabilities in applications,

accessing users‘ sensitive data and Granting permissions that they should not have been granted are also be discussed in Android applications for many years. Table 4 also shows the permission needed by each categories.

Table 3. Probabilities of each vulnerability category

Vulnerability category	Probability
A2. Cryptographic Failures	97.15%
A3. Code injection	66.36%
A4. Insecure Design	91.00%
A6. Vulnerable and Outdated Components	98.77%
A7. Identification and Authentication Failures	90.68%
A8. Software and Data Integrity Failures	82.84%

Table 4. Permissions needed by each category

PERMISSION	APPLICATION CATEGORY			
	A	B	C	D
INTERNET	100%	100%	100%	100%
CAMERA	92%	70%	92%	90%
WRITE TO EXTERNAL STORAGE	92%	70%	75%	80%
ACCESS TO THE NETWORK STATE	84%	90%	84%	60%
FINGERPRINT ACCESSSION	44%	50%	48%	50%
REQUEST ACCOUNT INFORMATION	32%	30%	34%	30%
WAKE UP LOCK	72%	70%	75%	70%
CLOUD MESSAGES	36%	50%	58%	60%
NFC	4%	0%	25%	10%
READ SMS	16%	10%	41%	50%
ACCESS TO THE BLUETOOTH	8%	10%	16%	20%
ACCESS TO THE GPS	24%	10%	8%	50%

According to the permission-based privacy analysis model in [30], all tested applications need the access permission to the Internet, which is reasonable because all cryptocurrencies need to make online transactions. Most of programs need to accsess camera and external storage because they might need to do QR code scan, take photo of band card information and record transactions. Most of access permissions seem to be reasonable and proper, but the Android OS needs to do more fine-grained management. For example, accessing camera is acceptable, but this should be done only when this particular application is used, when the screen is locked, the camera should not be allowed to be opened by the processes running in the background. Moreover, applications should not be allowed to access all senstive data in SMS, especially information which have no relation to them and should not be permitted to monitor user‘s network traffic.

5. Conclusion

The prosperity of cryptocurrencies shows the vitality and potential of this emerging field. However, the growth of every field should not be disordered. All analyses, conclusions and recommendations in this paper are not aim to stop the development of this industry. On the contrary, secure cryptocurrency applications make cryptocurrency be beneficial for all human beings. Based on analyses and conclusion made above, the suggestions of this paper are as follow: The development of an industry needs cooperation of every parts. Application programmers should have some formulates or published standards to follow. People had better holding conference regularly, where experts and scholars discuss emerging loopholes, out-dated components, applications not meeting the requirements. App stores should also label vulnerable applications and take down malwares disguised as normal cryptocurrency applications. Users are supposed to be careful when they first install one

application on their cell phones and report attacks they have confronted with to app store administrators. Moreover, regulations and laws play a significant role in this field. Malicious application programmers, privacy stealer and other attackers should be punished and arrested. According to and, plagiarisms in cryptocurrency apps is not rare. Plagiarism violates intellectual property laws in the first place. If plagiarists do not receive penalty, how could other publishers be willing to come up with novel ideas and update versions of original applications. In addtion, most of plagiarists do not know deeply about their application, they stand a high chance no longer maintaining their applications as well. Punishing these plagiarists and attackers can only be done by goverments and laws. Decentralization is an advantage of digital economy. In blockchain, no single person or a selected group controls the blockchain network, but the power is distributed at every end point. Most of sites need to reach agreement before a sub-chain enter the whole network. Introducing this mechanism into community management is also important and feasible. If given portition of people inside the community agree a specific application is not suitable to stay inside this community, it should be evited. Communities play the same role as certificate authorities and provide credit endorsement for applications inside the community. Users could download and use applications inside those communities they trusted.

References

[1] Fang F, Ventre C, Basios M, et al. Cryptocurrency trading: a comprehensive survey[J]. Financial Innovation, 2022, 8(1): 1-59.

[2] Liu Y, Tsyvinski A, Wu X. Common risk factors in cryptocurrency[J]. The Journal of Finance, 2022, 77 (2): 1133-1177.

[3] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. Decentralized Business Review, 2008: 21260.

[4] Atzeni A, Diaz F, Lopez F, et al. The rise of android banking trojans[J]. IEEE Potentials, 2020, 39(3): 13-18.

[5] Priyawati D, Rokhmah S, Utomo I C. Website Vulnerability Testing and Analysis of Website Application Using OWASP[J]. International Journal of Computer and Information System (IJCIS), 2022, 3(3): 142-147.

[6] Shi Y, Zhang Y, Luo T, et al. Backporting Security Patches of Web Applications: A Prototype Design and Implementation on Injection Vulnerability Patches[C]//31th USENIX Security Symposium (USENIX Security). 2022.

[7] Banks J. The Heartbleed bug: Insecurity repackaged, rebranded and resold[J]. Crime, Media, Culture, 2015, 11(3): 259-279.

[8] Ordean M, Giurgiu M. Towards securing client-server connections against man-in-the-middle attacks [C] // 2012 10th International Symposium on Electronics and Telecommunications. IEEE, 2012: 127-130.

[9] Sterle L, Bhunia S. On SolarWinds Orion Platform Security Breach[C]//2021 IEEE Smart World, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/ SCALCOM/ UIC/ATC/ IOP/ SCI). IEEE, 2021: 636-641.

[10] Pellegrino G, Catakoglu O, Balzarotti D, et al. Uses and abuses of server-side requests[C]//International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, Cham, 2016: 393-414.

[11] Zhang X, Wang X, Slavin R, et al. Condysta: Context-aware dynamic supplement to static taint analysis [C]// 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 2021: 796-812.

[12] Lu L, Li Z, Wu Z, et al. Chex: statically vetting android apps for component hijacking vulnerabilities [C]//Proceedings of the 2012 ACM conference on Computer and communications security. 2012: 229-240.

[13] Bosu A, Liu F, Yao D, et al. Collusive data leak and more: Large-scale threat analysis of inter-app communications[C]//Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. 2017: 71-85.

[14] Wei F, Roy S, Ou X. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps[J]. ACM Transactions on Privacy and Security (TOPS), 2018, 21(3): 1-32.

[15] Hu G, Zhang B, Xiao X, et al. SAMLDroid: A Static Taint Analysis and Machine Learning Combined High-Accuracy Method for Identifying Android Apps with Location Privacy Leakage Risks[J]. Entropy, 2021, 23(11): 1489.

[16] Usui T, Otsuki Y, Kawakoya Y, et al. Script Tainting Was Doomed from The Start (By Type Conversion): Converting Script Engines into Dynamic Taint Analysis Frameworks[C]//Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses. 2022: 380-394.

[17] Loch F D, Johns M, Hecker M, et al. Hybrid taint analysis for java ee[C]//Proceedings of the 35th Annual ACM Symposium on Applied Computing. 2020: 1716-1725.

[18] Gordon M I, Kim D, Perkins J H, et al. Information flow analysis of android applications in droid safe [C] // NDSS. 2015, 15(201): 110.

[19] Arzt S, Rasthofer S, Fritz C, et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps[J]. Acm Sigplan Notices, 2014, 49(6): 259-269.

[20] Wei F, Roy S, Ou X. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps[J]. ACM Transactions on Privacy and Security (TOPS), 2018, 21(3): 1-32.

[21] Desnos A, Gueguen G. Android: From reversing to decompilation[J]. Proc. of Black Hat Abu Dhabi, 2011, 1: 1.

[22] Specification P F. Tool Interface Standard (TIS) Portable Formats Specification[J]. 1993.

[23] Wang X, Yu H. How to break MD5 and other hash functions[C]//Annual international conference on the theory and applications of cryptographic techniques. Springer, Berlin, Heidelberg, 2005: 19-35.

[24] Bošnjak L, Sreš J, Brumen B. Brute-force and dictionary attack on hashed real-world passwords[C]//2018 41st international convention on information and communication technology, electronics and microelectronics (mipro). IEEE, 2018: 1161-1166.

[25] Kumar H, Kumar S, Joseph R, et al. Rainbow table to crack password using MD5 hashing algorithm [C]// 2013 IEEE Conference on Information & Communication Technologies. IEEE, 2013: 433-439.

[26] Kausar S, Tahir B, Mehmood M A. HashCat: A Novel Approach for the Topic Classification of Multilingual Twitter Trends[C]//2021 International Conference on Frontiers of Information Technology (FIT). IEEE, 2021: 212-217.

[27] Zhu X, Xu H, Zhao Z, et al. An Environmental Intrusion Detection Technology Based on WiFi[J]. Wireless Personal Communications, 2021, 119(2): 1425-1436.

[28] Sai A R, Buckley J, Le Gear A. Privacy and security analysis of cryptocurrency mobile applications [C]// 2019 fifth conference on mobile and secure services (mobisecserv). IEEE, 2019: 1-6.

[29] Fang Z, Han W, Li Y. Permission based Android security: Issues and countermeasures[J]. computers & security, 2014, 43: 205-218.

[30] Purba R, Yunis R. Application of Blockchain technology to prevent the potential of plagiarism in scientific publication[C]//2019 Fourth International Conference on Informatics and Computing (ICIC). IEEE, 2019: 1-5.

[31] Choi W, Kim H. How to Measure Similarity between Source Codes of Cryptocurrencies for Detecting Plagiarism (Lightning Talk) [C]//2018 International Conference on Software Security and Assurance (ICSSA). IEEE, 2018: 91-91.