 CMC, vol.59, no.1, pp.181-198, 2019

CM-Droid: Secure Container for Android Password Misuse Vulnerability

Wen Zhang[1], Keyue Li[1,*], Tianyang Li[1], Shaozhang Niu[1] and Zhenguang Gao[2]

Abstract: Android applications are associated with a large amount of sensitive data, therefore application developers use encryption algorithms to provide user data encryption, authentication and data integrity protection. However, application developers do not have the knowledge of cryptography, thus the cryptographic algorithm may not be used correctly. As a result, security vulnerabilities are generated. Based on the previous studies, this paper summarizes the characteristics of password misuse vulnerability of Android application software, establishes an evaluation model to rate the security level of the risk of password misuse vulnerability and develops a repair strategy for password misuse vulnerability. And on this basis, this paper designs and implements a secure container for Android application software password misuse vulnerability: CM-Droid.

Keywords: Password misuse, evaluation model, secure container, dynamic repair.

1 Introduction

With the development of mobile Internet and the popularity of smart phones, smart phones have become an indispensable part of most people's lives. According to the smart phone system share report released by market research organization Gartner [Wang (2017)], in the first quarter of 2017, the sales of Android devices reached 327 million units, and the corresponding Android system share reached 86.1%, an increase of 2% over the same period last year. Since the number of applications in the Google Play app store exceeded 1 million in 2015, it has maintained a rapid growth trend. By January 2017, the number of applications has increased to 2.7 million.

In order to provide users with more software features, the Android system provides developers with interfaces such as reading geographic information, contact lists and other private data. In addition, users often need to input personal information such as account number and password in the process of using application software, which causes Android applications associating a large amount of user sensitive data. Developers use encryption algorithms to provide data encryption, authentication and integrity protection in

[1] Beijing Key Lab of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[2] Department of Computer Science, Framingham State University, 100 State St, Framingham, Massachusetts, MA 01702, USA.

[*] Corresponding Author: Keyue Li. Email: likeyue@bupt.edu.cn.

applications. However, in the analysis report of Veracode [Danhieux (2016)] in 2015, 87% of Android applications have encryption problems, which is 36% higher than the 51% in 2013. In the 2016 Annual Report published by Aliju Security, the Top10 Android application software of 18 industries in 2016 was analyzed. Among the 14798 bugs detected, 3174 (21.4%) were password misuse loopholes. This shows the phenomenon of password misuse is very common in Android applications.

This paper summarizes the characteristics of password misuse vulnerability of Android application software, establishes an evaluation model to rate the security level of the risk of password misuse vulnerability and develops a repair strategy for password misuse vulnerability. And on this basis, this paper designs and implements a secure container for Android application software password misuse vulnerability: CM-Droid. The container is able to quickly locate password misuse vulnerabilities in Android apps and fixes vulnerabilities through flexible security policies.

2 Previous work

In the early stages, the research on password misuse vulnerability of Android platform is mainly concentrated in the direction of SSL/TLS protocol. In 2008, Bhargavan et al. [Bhargavan, Fournet, Corin et al. (2008)] showed how to use detection tools to analyze the security of the use process of cryptographic protocols. In 2012, Georgiev et al. [Georgiev, Iyengar, Jana et al. (2012)] used a man-in-the-middle attack to detect whether an SSL security vulnerability exists in the application. In 2014, Sounthiraraj and others [Sounthiraraj, Sahs, Greenwood et al. (2014)] proposed the SMV-HUNTER system, which could automatically identifies SSL/TLS man-in-the-middle attacks vulnerability for large-scale Android applications. In 2015, Onwuzurike [Onwuzurike and De Cristofaro (2015)] conducted research on information leakage and SSL vulnerabilities in Android applications. However, the above work only studies the password misuse problem of the protocol, and does not systematically detect the password misuse vulnerability existing in the entire application software. In 2013, Egele et al. [Egele, Brumley, Fratantonio et al. (2013)] systematically analyzed the password misuse vulnerability of Android applications. In 2014, Shao et al. [Shao, Dong, Guo et al. (2014)] established a password misuse detection system CMA, which uses a combination of dynamic and static methods to detect password misuse vulnerability in Android programs. In 2015, Chatzikonstantinou et al. [Chatzikonstantinou, Ntantogian and Karopoulos (2015)] chose to use weak encryption, weak implementation, weak key and weak encryption parameters as detection items to detect Android application password misuse vulnerability. In 2018, Li et al. [Li, Luo, Zhao et al. (2018)] proposed a provably secure APK redevelopment authorization scheme in the standard model.

In addition, González et al. [González, Esparza, Muñoz et al. (2015)] analyzed the encryption algorithms and encryption structures provided by Android in the research. They found that the encryption algorithms provided were not the same in multiple system versions of Android. Somak et al. [Somak, Gopal, King et al. (2014)] studied encryption libraries including programming languages such as C, C++, Java, Python and Go. However, the above research only detects whether the application has a password misuse security problem, and cannot fix the existing vulnerability in the program. Ma et al. [Ma,

Lo, Li et al. (2016)] implemented an automatic repair tool CDRep for Android application password misuse vulnerability. But there are two problems with this scheme: 1. Only use static detection to detect password misuse vulnerabilities in the application, so there may be a false positive. 2. The re-packaging method is used to complete the bug fix. This method cannot be applied to the application software with anti-repackaging function. It can be seen that there is still a significant gap in the security protection research for Android application password misuse vulnerability, which is also the focus of this paper.

3 Android application password misuse vulnerability analysis

3.1 Analysis of the current situation of Android software password usage

We downloaded Top30 applications in the five categories of wealth management, communication, music, entertainment, and reading from the Yingyongbao app store, and then analyzed the use of encryption algorithms for these applications. The analysis results are shown in Tab. 1. Through analysis of these 30 APPs, we found that the category with the highest ratio of using encryption algorithms is the reading class, which has achieved 57% usage. In addition, most applications that use encryption algorithms use three types of encryption algorithms at the same time. There are few applications that use only one or two types of algorithms. This illustrates the need to detect password misuse vulnerabilities in Android apps from multiple dimensions.

Table 1: Statistics table of the usage of encryption algorithm in various applications

Application category	Amount	Symmetric encryption algorithm	Asymmetric encryption algorithm	Hash encryption algorithm
wealth management	15	15	15	15
communication	13	12	13	13
news	15	14	14	15
music	13	13	13	12
reading	17	14	15	17
total	73	68	70	72

3.2 Android password misuse vulnerability risk rating

3.2.1 Quantitative analysis of the risks

For the classification of application software password misuse, CWE [CWE (2018)] lists more than 20 security vulnerabilities that developers need to pay attention to during the application development process, but the CWE classification is not specifically analyzed for the Android platform. Based on the CWE classification, we also refer to the method of Shuai [Shao, Dong, Guo et al. (2014)], and divide the password misuse vulnerability of Android application software into four categories: symmetric encryption algorithm class, asymmetric encryption algorithm class, hash algorithm class and password management class. We quantify the security impact of these vulnerabilities. For the convenience of explanation, in the following analysis process, We use m, n for the length of the

encryption key, l for the output length of the hash function, and k for the constant,and there are m, n, l, k∈N, m<n, 0<k<n. See Appendix 1, where L(n) stands for

$$exp((\sqrt[3]{\frac{64}{9}}+O(1))(\ln n)^{\frac{1}{3}}(\ln\ln m)^{\frac{2}{3}}) \quad (1)$$

Symmetric encryption algorithm (SE): symmetric encryption algorithms provided by the Android platform include DES, AES, etc. Symmetric encryption algorithm class vulnerabilities include three subclasses: encryption mode misuse, initial vector misuse, and encryption algorithm misuse.

Asymmetric Encryption Algorithm (ASE): The asymmetric encryption algo-rithm provided by the Android platform is mainly the RSA algorithm. The vulnerabilities of asymmetric encryption algorithms include four subclasses: key length misuse, low decryption index misuse, padding mode misuse, and certificate validation vulnerabilities.

Hash Algorithm (HA): The hash algorithm is a one-way algorithm. The user can generate a unique hash value of a specific length by using a hash algorithm. Hash algorithm class vulnerabilities include two subclasses: cryptographic algorithm misuse and input parameter misuse. Tab. 4 lists the quantitative analysis of vulnerabilities.

Key Management (KM): A key management class vulnerability is when an application software stores or passes an encryption key in an unsecure manner. Key management class vulnerabilities mainly include key storage vulnerabilities and key generation vulnerabilities.

3.2.2 Threat rating

According to the research results, the attack complexity of the encryption algorithm can be divided into five categories: constant order, linear order, polynomial order, sub-exponential order, and exponential order according to the order of magnitude. This article uses M (magnitude) to represent the order of magnitude of complexity: M1 represents a constant order and the attack complexity belonging to M1 is O(1); M2 represents a linear order and the attack complexity belonging to M2 is O(n); M3 represents a polynomial order, and the attack complexity belonging to M3 is O(p(n)); M4 represents the sub-exponential order, and the attack complexity belonging to M4 is O(L(n)); M5 represents the exponential order, and the attack complexity belonging to M5 is O(2n/2) and O(2n). We classify the security threat level of password misuse vulnerability into three levels, namely high risk, medium risk and low risk. The criteria for judging are shown in Tab. 2.

Table 2: Security threat level standard

Threat level	Complexity change	Description
High risk	[M_5, M_2], [M_5, M_1]	The attack complexity is reduced from exponential to linear or constant.
Intermediate risk	[M_5, M_3]	The attack complexity is reduced from exponential to polynomial.
Low risk	[M_5, M_5], [M_5, M_4]	The attack complexity is reduced from exponential to exponential or sub-exponential.

At low-risk levels, the complexity of the vulnerability is of the same order of magnitude, or is reduced from an exponential step to a sub-exponential order. Attackers use the vulnerability to gain less benefit, and the attack process still requires complex calculations or takes a long time, so such vulnerabilities are relatively less threatening to the application.

At the medium-risk level, the attack complexity after exploiting the vulnerability is a polynomial order. The use of such vulnerabilities can significantly reduce the amount of attackers' calculations, and may even cause attacks that previously took several months to complete in a relatively short and acceptable time, so such vulnerabilities pose a significant hazard to the application.

At high risk levels, the attack complexity after exploiting the vulnerability is reduced to a linear order or a constant order. That is, the algorithm itself is very secure, but after the vulnerability occurs, the attacker can use the external factors in the algorithm to complete the attack very easily. Such vulnerability seriously affects the security of the application.

Based on the above definition of security threat levels for password misuse vulnerabilities, we rated and summarized the security risks of various types of password misuse vulnerabilities. The specific rating results are shown in Tab. 3.

Table 3: Password misuse security threat rating

Vulnerability number	Quantization formula	Complexity change	Threat level
SE-MM	SE-MM-[$O(2^n)$, $O(n)$]	[M_5, M_2]	High risk
SE-IM	SE-IM-[$O(2^n)$, $O(n)$]	[M_5, M_2]	High risk
SE-AM	SE-AM-[$O(2^n)$, $O(2^{n-k})$]	[M_5, M_5]	low risk
ASE-KLM	ASE-KLM-[$O(2^{n/2})$, $L(n)$]	[M_5, M_4]	low risk
ASE-LDEM	ASE-LDEM-[$O(2^{n/2})$, $O(p(n))$]	[M_5, M_3]	intermediate risk
ASE-PM	ASE-PM-[$O(2^{n/2})$, $O(n)$]	[M_5, M_2]	High risk
ASE-CVV	ASE-CVV-[$O(2^{n/2})$, $O(1)$]	[M_5, M_1]	High risk
HA-AM	HA-AM-[$O(2^{l/2})$, $O(2^{(l-k)/2})$]	[M_5, M_5]	low risk
HA-IM	HA-IM-[$O(2^{l/2})$, $O(2^{(l-k)/2})$]	[M_5, M_5]	low risk
KM-KSV	KM-KSV-[$O(2^n)$ /$O(2^{n/2})$, $O(1)$]	[M_5, M_1]	High risk
KM-KGV	KM-KGV-[$O(2^n)$/ $O(2^{n/2})$, $O(1)$]	[M_5, M_1]	High risk

3.3 Android password misuse vulnerability repair model

3.3.1 Password misuse causes

According to the principle of password misuse vulnerability, we summarize the reasons for the password misuse vulnerability in the development process. We abstract three main causes, algorithm selection errors, algorithm parameter setting errors, and password usage process errors, as described below:

R1, the algorithm selection is wrong. It mainly means that when using the cryptographic algorithm, the developer selects a cryptographic algorithm that has been proven to be unsafe.

R2, the algorithm parameter setting is wrong. It mainly means that when the password algorithm is used by the developer, the wrong password parameter is set or the password parameter is generated in an unsafe manner.

R3, the key usage process is wrong. It mainly means that when the cryptographic algorithm is used by the developer, there are obvious problems in the process of generating, storing, and transmitting the key, which causes the attacker to obtain the key under a very low attack condition.

3.3.2 Repair strategy

This article divides the application data into two categories according to the scope of the encrypted data, namely local data (L) and shared data (S). Local data refers to data stored only in the local sandbox or external storage area and used only within the application, which support the local business needs of the application software, but will not leave the scope covered by the application software. Shared data refers to data that is shared with external systems by applications such as inter-process communication, file sharing, and network communication. According to the cause of the Android password misuse vulnerability and the data type of the application software, this paper proposes five repair strategies: parameter fix, storage fix, key fix, block fix, and warning fix.

Parameter Fix (PF): When there are R1 and R2 password misuse holes in the encryption and decryption process, the vulnerability is replaced by replacing the calling parameters of the encryption API to avoid security risks. Since the repair strategy modifies key factors such as encryption algorithms and algorithm parameters in the data encryption process, it will affect the decryption process of the data. This policy can only fix password misuse holes in local data when it is executed independently.

Storage Fix (SF): When there is a password misuse vulnerability of the R3 class key storage problem during the encryption and decryption process, the vulnerability is repaired by modifying the storage path of the data to avoid security risks. This fix policy fixes the vulnerability mainly by changing the access rights of the local key file, so it only applies to the password misuse vulnerability of local data.

Key Fix (KF): When there is a password misuse vulnerability of the R3 class key generation problem during the encryption and decryption process, the vulnerability repair is completed by modifying the key generation method to avoid the occurrence of security risks.

Block Fix (BF): When there is a high-risk password misuse vulnerability in the encryption and decryption process, the vulnerability is fixed by forcibly blocking the encryption and decryption process of the data to avoid the occurrence of security risks. This repair strategy can be applied to password misuse vulnerabilities of local data and shared data, but it will affect the data execution logic of the application software, thus requiring user authorization and licensing.

Warning Fix (WF): When there is a password misuse vulnerability in the encryption and decryption process, by prompting the user to risk, the user actively chooses to stop the business logic to fix the vulnerability and avoid the occurrence of security risks. This repair strategy can also be applied to password misuse vulnerabilities of local data and shared data, but does not hinder the original business logic throughout the process.

3.3.3 CFMM model

CMFM adopts three repair strategies: parameter fix, storage fix and key fix. It will securely repair the encryption algorithm, encryption mode, encryption parameters, stored procedure, and key generation process used in the encryption and decryption process. CMFM abstracts the repair strategy of each vulnerability by repairing the vector-based four-dimensional tuple, including vulnerability number, vulnerability cause, encrypted data type, and repair strategy. Among them, each dimension element represents the key elements in the process of policy instantiation, as shown in Fig. 1.

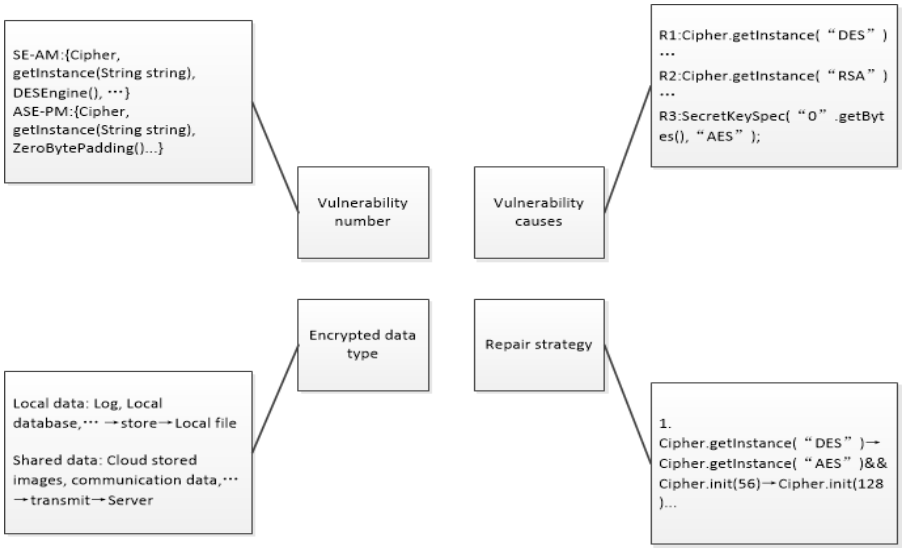

Figure 1: Four repair strategies

Therefore, the repair of the password misuse vulnerability through CMFM can make the encryption and decryption process in the application have no vulnerability of constant order and linear order attack complexity, effectively avoid the security risk introduced by the password misuse vulnerability, and greatly improve the security of data in the application.

4 Design of CM-Droid

Based on the repair model set in the previous chapter, this chapter proposes a secure container based on the application layer: CM-Droid. The container performs real-time monitoring on the execution process of the Android application invoking the encryption library, and through the flexible reinforcement policy configuration, the password misuse vulnerability of the application software is repaired to prevent the attacker from exploiting the password misuse vulnerability and prevent implementing key extraction, man-in-the-middle attacks, and encrypted data cracking of application software.

4.1 System architecture

The system architecture of CM-Droid consists of two parts, the server and the client. The server is responsible for analyzing the static code of the application and completing the

generation, storage and management of the policy; the client is the main body of the CM-Droid, which implements the secure running environment of the application, and is responsible for monitoring the encryption API and The repair of the vulnerability. The system architecture of CM-Droid is shown in Fig. 2.

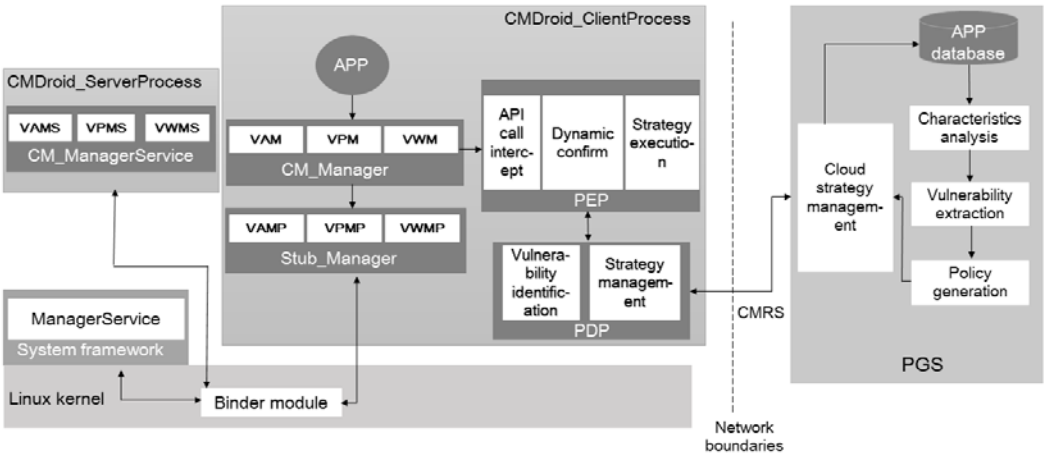

Figure 2: The system architecture of CM-Droid

PGS: PGS (Policy Generation Server) is the first module to be executed to prepare for the dynamic execution of the client. The PGS module first performs static code analysis on the APP uploaded by the client, including basic feature analysis, deep feature analysis, and vulnerability extraction.

SRE: SRE (Safe Runtime Environment) is the basic building block of the entire container, simulating the operating environment of the system to ensure that the application runs in a safe environment.

PEP: PEP (Policy Enforcement Point) is responsible for monitoring the behavior of the application runtime, confirming the suspected vulnerability, and repairing the encryption and decryption process of the confirmed vulnerability in real time.

After receiving the result returned by the PDP, the PEP will perform the corresponding operation according to the detection result. The PEP will complete the hardening according to the repair policy returned by the PDP.

PDP: The main role of the PDP (Policy Decision Point) is to perform security diagnosis on the current operation.

4.2 Key processes

4.2.1 Strategy generation

Policy generation is performed on the PGS (Policy Generation Server) remote server. It mainly analyzes the application and generates CMRS files. CMRS is used for Section 4.3 dynamic validation and Section 4.4 bug fixes.

The strategy generation mainly includes the following four stages, basic feature

extraction, depth feature analysis, dynamic analysis preprocessing and policy generation, as shown in Fig. 3.

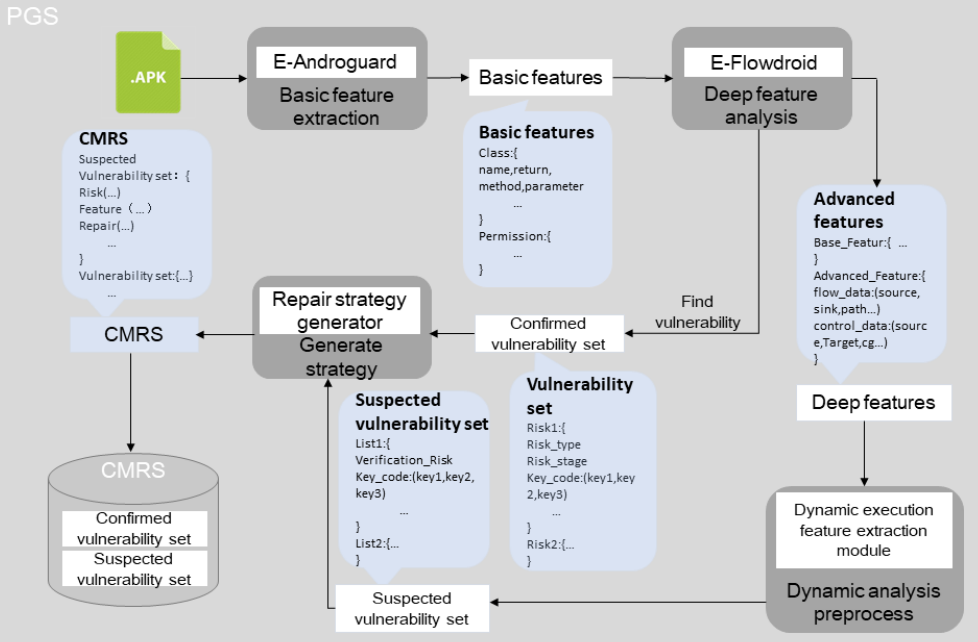

Figure 3: Strategy generation

Basic feature extraction: In PGS, we analyze the decompiled intermediate files by using the basic parsing of the APK file and combining Androguard to obtain the basic features of the application software.

Depth feature preprocessing information: In addition to the basic information of Apk, key API information, we also need to extract the information needed for deep feature analysis.

Depth feature analysis: Based on the basic features, we obtained some infor-mation, such as SE-AM. According to the code fragment const-string/jumbo v2, we can know that some insecure DES algorithm is used in the code. But it can't be linked to the vulnerability fix point we need: Ljava/crypto/Cipher;-> getInstance (encryption algorithm, working mode, filling method). We need to perform control flow analysis on the code, creating a path from the code fragment const-string/jumbo v2 to the bug fix point Ljava/crypto/Cipher;-> getInstance(). And record the vulnerability fix point.

After basic feature extraction and depth feature analysis, we combine the collected code features with the definition of the vulnerability to generate a validation vulnerability set, confirming that the vulnerability set contains the vulnerability name, description, fix point features, and fix point location.

Dynamic Analysis Preprocessing: Some vulnerabilities we cannot or are difficult to confirm with static analysis. For example, key length misuse (SE-KLM), the repair point feature is, Ljavax/crypto/KeyGenerator; -> initialize(I)V. The basic data type in the code

is difficult to get the actual value, and the actual value of the parameter can only be obtained when the application is running.

Strategy generation: In the third chapter, we have detailed descriptions of the vulnerability repair strategy, and we will not make a detailed statement here. Based on the name of the vulnerability, we generate the corresponding repair strategy and finally generate a CMRS file.

4.2.2 Dynamic confirmation

The dynamic confirmation of the suspected vulnerability of CM-Droid mainly involves vulnerabilities in the dynamic execution of suspected vulnerabilities in the policy file to determine whether the vulnerability exists. Participants in the entire process include the remote PGS and the local CM-Droid container. The dynamic confirmation process for suspected vulnerabilities is shown in Fig. 4.

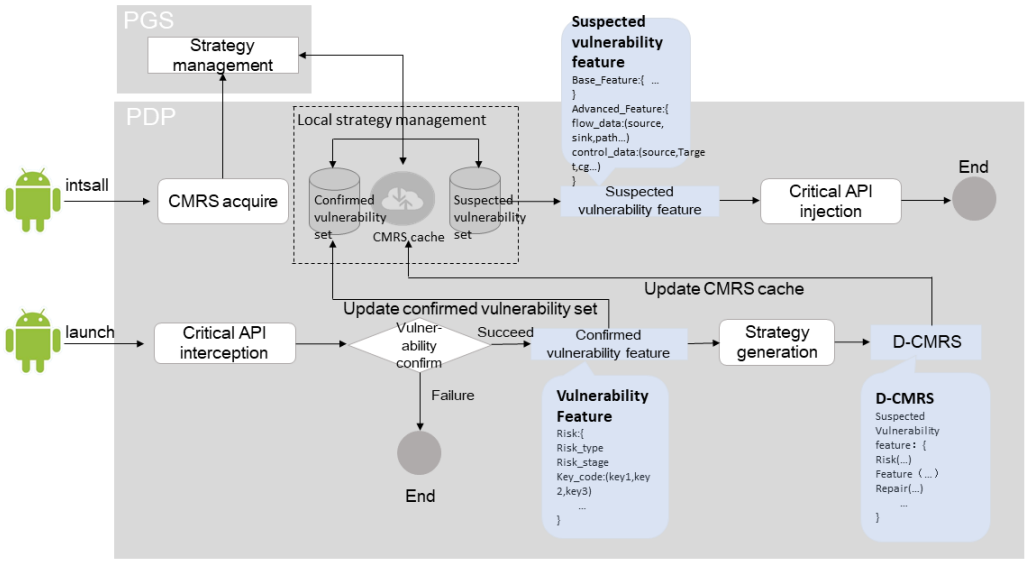

Figure 4: Dynamic confirmation

Install phase: After the application is installed into the container, the PDP module will first extract the basic information of the application: Apk_name, Apk_Version, Apk_Size. Communicate with the PGS to obtain the CMRS file of the application from the PGS. PGS first looks for basic information based on the application. If the remote library contains eligible CMRS, it will return this CMRS directly. If the Apk does not exist in the PGS remote library, the Apk is transmitted remotely, and the policy generation process is executed.

Launch phase: After the application is launched, when it is executed to the key API, it will be intercepted by the PEP module. The PEP module will collect the feature package of the Apk at this time: the API of the hook, the API parameters, and the stack information. The feature is sent to the PDP module, and the PDP module determines the vulnerability.

Verification method: In CM-Droid, the dynamically confirmed vulnerabilities include ASE-CVV, ASE-LDEM, HA-IM, KM-KSV, etc. These vulnerabilities generate conditions during the policy generation phase, and we validate the condition. If the condition is met, the suspected vulnerability is verified and a confirmed vulnerability set is added.

4.2.3 Vulnerabilities repair

CM-Droid's bug fixes are mainly for the vulnerability of the APP that has been confirmed in the local policy cache, and are dynamically repaired during the running of the APP. The entire repair process is divided into two phases, as shown in the Fig. 5

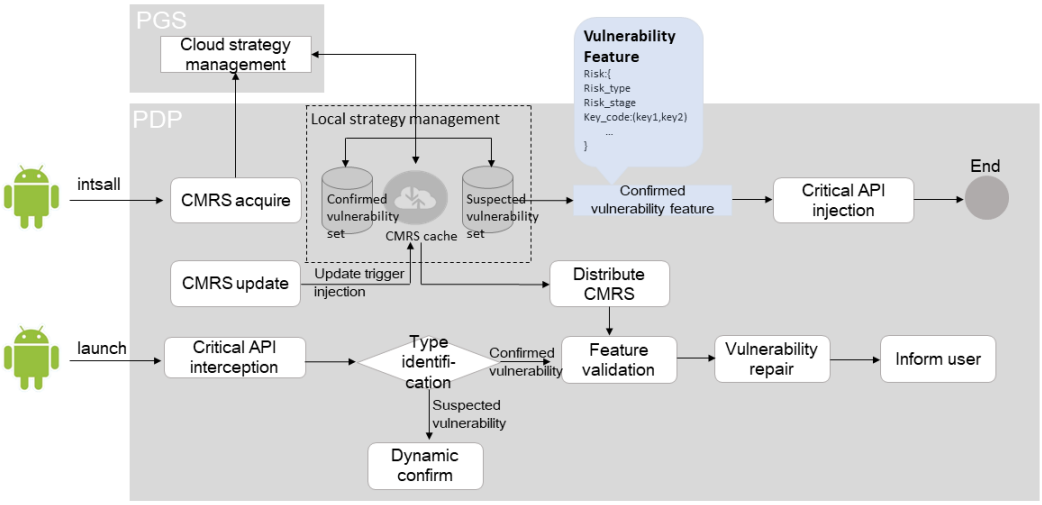

Figure 5: Vulnerabilities repair

Install phase: In the installation phase of vulnerability confirmation, CM-Droid obtains the CMRS file by PGS or CMRS gained by dynamically confirmed triggered updated. The PDP module extracts the acknowledgment vulnerability feature from the acknowledgment vulnerability set and sends it to the PEP module. After receiving the acknowledgment vulnerability feature, the PEP module extracts the key API and injects it.

Launch phase: After the application is launched, it will be intercepted by the PEP module when it executes the key API with the confirmation vulnerability. The PEP module will determine whether the acquired information matches the vulnerability feature. Once the match is successful, the PEP requests the PDP module for the repair strategy corresponding to the vulnerability. The PDP sends a corresponding repair policy according to the vulnerability information, and the PEP module receives the repair policy and performs the vulnerability repair operation.

5 Experiment and evaluation

This chapter mainly tests and evaluates the CM-Droid-based Android client vulnerability repair tool designed in this paper, and verifies the accuracy and effectiveness of the tool

for the repair of password misuse. This chapter firstly deploys the test environment, and then performs functional tests on CM-Droid. It mainly verifies whether the functions of CM-Droid can run normally, and then tests the performance of CM-Droid. Finally, the conclusions of the experimental analysis are obtained, and suggestions for improving performance are proposed.

5.1 Test process design

This section mainly introduces the design of the test process for CM-Droid, including the purpose of the test, the configuration of the test environment, the selection of test objects and the test method.

5.1.1 Test purpose

The purpose of this test is first to verify the feasibility of the CM-Droid, and then to evaluate some of the performance indicators of the test tool in the actual detection of the vulnerability.

5.1.2 Test environment

The CM-Droid designed in this paper can be divided into two parts, namely the static vulnerability scanning part on the server side and the dynamic behavior analysis part in the client side. In this test, the static vulnerability scanning part was deployed on the 0.4 version of the Santoku system.

5.1.3 Test method

In order to verify the repair effect of CM-Droid, we detected and repaired 15 popular applications through CM-Droid, manually analyzed the problems of 15 popular applications, and recorded the time and performance of CM-Droid detection.

5.2 Test results and analysis

5.2.1 Function test

We focus on testing the actual detection and repair capabilities of CM-Droid for 12 types of password misuse vulnerabilities. We focus on 15 applications on the market, through PGS-side policy generation, client-side dynamic validation and bug fixes. We have established the following experimental data:

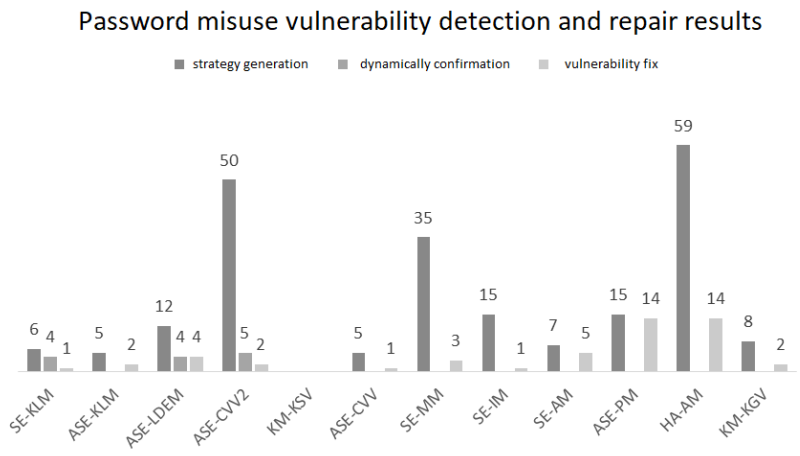

Figure 6: Password misuse vulnerability detection and repair results

For the experimental results, we performed a manual analysis of the missing KM-KSV vulnerability. After analysis, we believe that the KM-KSV vulnerability does not exist in popular applications. Popular apps don't store keys locally. At the same time, we analyze the ASE-CVV2 vulnerability. According to the experimental results, our tool CM-Droid can effectively detect the password misuse vulnerability in the application. At the same time, due to the triggering process, we actually fix fewer vulnerabilities than the detected vulnerability.

5.2.2 Repair time assessment

Our main time overhead is fixed in the CM-Droid container for password misuse vulnerabilities. So we perform a time performance assessment for the bug fix process.

For the performance evaluation of time, we first evaluate the extra time taken by the CM-Droid container for the repair application. We mainly test the time it takes for each type to go from repairing the trigger point to repairing the entire process. We tested each type of vulnerability for 1,000 times.

According to our definition of the repair strategy, the repair methods can be divided into replacement repair (parameter repair (PF), storage repair (SF), key repair (KF)), block repair (BF), and early warning repair (WF). We will compare the time between the vulnerabilities that can be used in these three types of repair methods.

Replacement fix: Due to the substitution of parameters, keys or storage locations, the repair time is related to the length of the encrypted data, so we use 12 bytes, 105 bytes, 1280 bytes of data for repair operations.

The replacement repair time has a linear relationship with the size of the encrypted data. In SE-AM, if the repair data is 1280 bytes, the program crashes directly. For practical reasons, if the encrypted data is too large, we will choose to block the repair and not replace it.

Blocking fix: When there is a high-risk password misuse vulnerability in the encryption and decryption process, the vulnerability is fixed by forcibly blocking the data encryption

and decryption process to avoid security risks. Due to the impact of the normal application process, we will pop up a dialog box, which is selected by the user.

Warning fix: When there is password misuse vulnerability in the encryption and decryption process, by prompting the user to risk, the user actively chooses to stop the business logic to fix the vulnerability and avoid the security risk. The warning fix here is that we only calculate the time from the discovery of the vulnerability to the pop-up warning. The user's choice time is not in our consideration.

The warning fix only prompts the user to make a selection, and the repair time is shorter. This does not hinder the original business logic, but requires the user to make a choice.

5.2.3 Performance evaluation

Our main performance overhead is for the launch and run of the application in the CM-Droid container. So we perform performance evaluations for the startup and operation of the application in the CM-Droid. We use the Android Profiler for performance evaluation. For performance evaluation, an application may have multiple processes.

The experimental results are shown in the Fig. 7, Fig. 8 and Fig. 9 respectively.

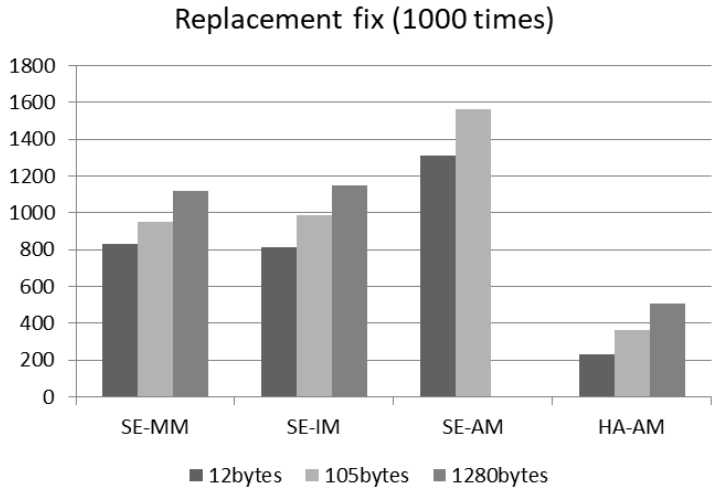

Figure 7: Replacement fix

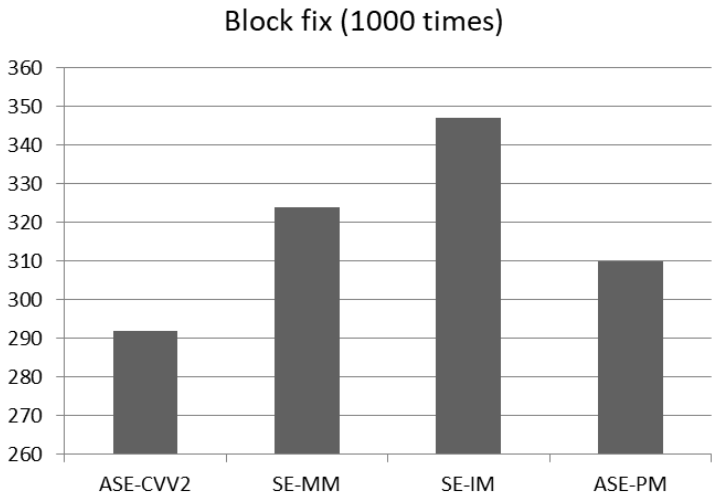

Figure 8: Block fix

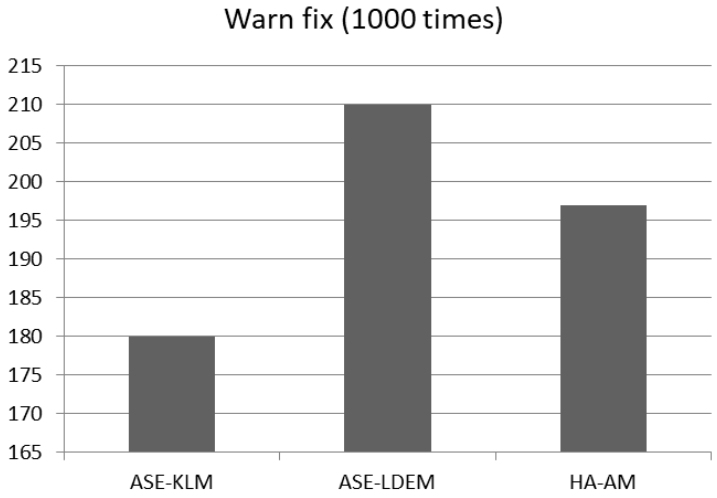

Figure 9: Warn fix

Memory evaluation: It is an evaluation of memory. We first evaluate the additional memory footprint of the CM-Droid container for repairing applications. We use the method of comparative testing. We tested the average memory and peak memory of the CM-Droid and compared the average memory and peak memory of the application when the CM-Droid is not running.

After calculation, the average memory running in the CM-Droid is 5.1% more than not running in the CM-Droid. The peak memory running in the CM-Droid is 7.3% more than not running in the CM-Droid. The extra memory is consumed in an acceptable range. The experimental results are shown in the Fig. 10.

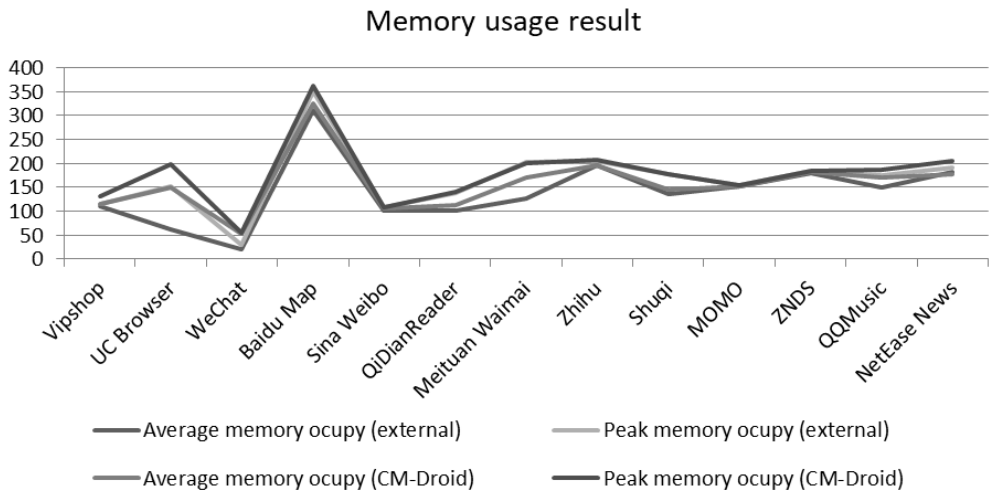

Figure 10: Memory usage result

CPU occupancy assessment: For the evaluation of CPU usage, we first evaluate the additional CPU usage that the CM-Droid container brings. We use the method of comparative testing. We tested the average CPU usage and peak CPU usage of the application in the CM-Droid and compare the average CPU usage and peak CPU usage of the application when they are not in the CM-Droid.

After calculation, the average CPU usage running in the CM-Droid is 26.4% more than not running in the CM-Droid. The peak CPU usage running in the CM-Droid is 10.1% more than not running in the CM-Droid. The CPU usage is relatively high, which is related to our simulation of system services. The experimental results are shown in the Fig. 11.

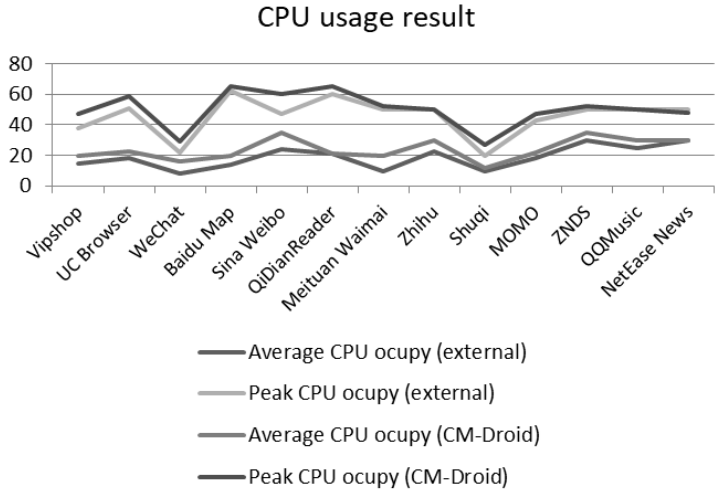

Figure 11: CPU usage result

6 Conclusion

This paper mainly summarizes the characteristics of password misuse vulnerability of Android application software, establishes an evaluation model to rate the security level of the risk of password misuse vulnerability and develops a repair strategy for password misuse vulnerability. And on this basis, this paper designs and implements a secure container for Android application software password misuse vulnerability: CM-Droid. Through experimentation, the additional time and memory loss increased by CM-Droid is within a acceptable range and achieves our predicted detection and repair results. But there are also some problems.

First of all, we are currently only targeting Android systems below 4.4 and 4.4. For Android systems above 4.4 version which use the ART virtual machine, we have not conducted research. In further work, we will implement CM-Droid for systems above 4.4. Secondly, the current PGS end detection has not been fully automated, there are still steps of manual detection and verification, and there are some errors. In next steps, we are going to achieve full automation of detection on PGS.

Acknowledgement: This work is supported by The National Natural Science Foundation of China Nos. U1536121, 61370195).

References

Bhargavan, K.; Fournet, C.; Corin, R.; Zalinescu, E. (2008): Cryptographically verified implementations for TLS. *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pp. 459-468.

Bleichenbacher, D. (1998): Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, vol. 1462, pp. 1-12.

Chatzikonstantinou, A.; Ntantogian, C.; Karopoulos, G. (2015): Evaluation of cryptography usage in Android applications. *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies*, vol. 3, no. 9, pp. 83-90.

Danhieux, P. (2016): Veracode's state of software security report supplement to vol 6, fall 2015: application development landscape. http://www.linkedin.com/pulse/veracodes-state-software-security-report-supplement-vol-danhieux.

Egele, M.; Brumley, D.; Fratantonio, Y.; Kruegel, C. (2013): An empirical study of cryptographic misuse in Android applications. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pp. 73-84.

Georgiev, M.; Iyengar, S.; Jana, S.; Anubhai R. (2012): The most dangerous code in the world: validating SSL certificates in non-browser software. *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pp. 38-49.

González, D.; Esparza, O.; Muñoz, J. L. (2015): Evaluation of cryptographic capabilities for the Android platform. *Future Network Systems and Security*, vol. 523, pp. 16-30.

Kunihiro, N.; Shinohara, N.; Izu, T. (2014): A unified framework for small secret exponent attack on RSA. *Proceedings of the 18th International Conference on Selected Areas in Cryptography*, vol. 7118, pp. 260-277.

Li, D.; Luo, M.; Zhao, B.; Che, X. (2018): Provably secure APK redevelopment authorization scheme in the standard model. *Computers, Materials & Continua*, vol. 56, no. 3, pp. 447-465.

Ma, S.; Lo, D.; Li, T.; Deng R. (2016): CDRep: automatic repair of cryptographic misuses in Android applications. *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 711-722.

Onwuzurike, L.; De Cristofaro, E. (2015): Danger is my middle name: experimenting with SSL vulnerabilities in Android apps. https://arxiv.org/abs/1505.00589.

Shao, S.; Dong, G.; Guo, T.; Yang, T. (2014): Modelling analysis and auto-detection of cryptographic misuse in Android applications. *IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pp. 75-80.

Somak, D.; Gopal, V.; King, K.; Venkatraman, A. (2014): IV=0 security cryptographic misuse of libraries. http://courses.csail.mit.edu/6.857/2014/files/18-das-gopal-king-venkatraman-IV-equals-zero-security.pdf.

Sounthiraraj, D.; Sahs, J.; Greenwood, G.; Lin, Z. (2014): SMV-HUNTER: large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in Android apps. *Network and Distributed System Security Symposium*.

Stevens, M.; Karpman, P.; Peyrin, T. (2016): Freestart collision for full SHA-1. *Advances in Cryptology-EUROCRYPT 2016 (35th Annual International Conference on the Theory and Applications of Cryptographic Techniques)*, vol. 9665, pp. 459-483.

Wang, Y. (2017): 2017 Q1 apple IOS system share decreased by 1.1% and Android rose 2%. http://mobile.zol.com.cn/640/6407752.html.