Development of a Reference Application for the OWASP Mobile Application Security Testing Guide (MASTG)

Bachelor's Thesis

Dominic Kronig

Universität Bern

Bern, January 30, 2026

First Reviewer: **Prof. Dr. Timo Kehrer**
Universität Bern
Institut für Informatik

Second Reviewer: **Thomas Sutter**
Universität Bern
Institut für Informatik

To my parents

Abstract: The Open Worldwide Application Security Project (OWASP) Mobile Application Security (MAS) project aims to define the industry standard for Android mobile application security. Its Mobile Application Security Testing Guide (MASTG) is supplemented with a catalog of reference apps. These contain intentionally built-in vulnerabilities for educational purposes and can furthermore be used for benchmarking security testing tools. However, many of the existing reference apps are limited in scope and have become outdated due to a lack of maintenance. We evaluated the state of the art in eight Android MASTG reference apps and found that only about a third of the vulnerabilities documented by OWASP MAS are represented in them. The ten most frequently implemented vulnerability types account for half of all implemented vulnerabilities. Based on the findings, we developed three open-source MASTG reference apps containing 28 vulnerabilities of both static and dynamic nature. We focused on vulnerabilities that received no coverage in existing reference apps. Each implemented vulnerability is mapped to its OWASP counterpart and contains detailed documentation on where it can be found. Our apps have increased the coverage of vulnerability types by 42% and can be used both as educational resources and for benchmarking security testing tools.

Contents

1. Introduction

1.1. Motivation

Mobile applications (apps) can have a variety of vulnerabilities. Modern Android apps have, on average, a surprisingly large number of vulnerabilities, which greatly impacts their security and usage [1]. Because of this, tools that can detect the vulnerabilities of mobile apps are of great importance. These tools can evaluate an app's security either statically, by analyzing the code, or dynamically, by executing the app and inspecting it at runtime. There exists a diverse landscape of such Static Security Application Testing (SAST) and Dynamic Security Application Testing (DAST) tools. But how can the effectiveness and accuracy of these tools be measured and compared to one another?

To measure the effectiveness of such a tool, it is important to know both how many security vulnerabilities the tool detected and how many it failed to find. Most real-world mobile apps are closed source. This makes them a poor dataset for benchmarking these security analysis tools, as there is no ground truth against which their performance on the apps can be compared. A better approach for effectively measuring and comparing the accuracy of existing analysis tools is to evaluate them on intentionally vulnerable apps, of which the existing vulnerabilities are known to the person conducting the tests.

1.2. Problem Statement

The OWASP MAS project hosts such a dataset of intentionally vulnerable apps that function as practical reference material for the MASTG [2]. They serve as a valuable educational resource that teaches mobile developers what security vulnerabilities can look like and how they can be found. These intentionally vulnerable apps are furthermore good resources for benchmarking mobile security tools.

Unfortunately, many of these MASTG reference apps have been designed for older versions of Android and have not received any updates for years. Because of this, they do not adequately reflect and implement the modern vulnerabilities that the MASTG tests for. This greatly diminishes their educational value as learning and teaching resources.

Very few of them contain documentation of their implemented vulnerabilities. Their usefulness for benchmarking security tools is negatively impacted because of this, as it makes it more difficult to assess how many vulnerabilities have not been found by the tools. All of this results in a gap between the educational value provided by the OWASP MAS project and how it is reflected in its MASTG reference apps, as well as the practical value it provides in mobile security testing and research.

1.3. Objectives of the Thesis

The objectives of this thesis mostly concern the development of multiple open-source OWASP MASTG reference apps that include both static and dynamic security vulnerabilities. To achieve this, the thesis pursues the following objectives:

1. Analyze and understand the OWASP MAS ecosystem and its vulnerability categories.

2. Conduct a literature review to better understand how the OWASP MASTG, its reference apps, as well as other related components, are used in the academic world.

3. Evaluate the state of the art in intentionally vulnerable OWASP MASTG Android reference apps in terms of vulnerability coverage, documentation, educational value, and usability for SAST and DAST tools benchmarking.

4. Design and implement new MASTG Android reference apps, adhering to the determined state-of-the-art standards. The focus lies on vulnerabilities that have seen little to no coverage in existing reference apps. Both static and dynamic vulnerabilities are to be included.

5. Document each implemented vulnerability by mapping it to its OWASP MAS counterpart, explaining where it can be found in the implementation's code and how it can be exploited.

6. Compare our reference apps with the existing reference apps landscape and assess the contribution made by our apps. Compare the coverage of vulnerabilities with and without our apps. Evaluate their suitability as teaching resources and for benchmarking security analysis tools.

1.4. Methodology and Structure of the Thesis

This thesis follows a design- and implementation-focused research approach. It consists of first reviewing and analyzing the existing work in the related field. Multiple pieces of software are then designed, implemented, and evaluated. We begin with an analysis of the OWASP MASTG and its reference applications. The choice of which vulnerabilities to implement is made based on the results of this analysis. After coming up with the design and architecture for our new reference apps, we shift the focus to their development. The resulting applications are then compared to the existing catalog of MASTG reference apps. They are evaluated for their coverage of vulnerabilities, as well as for their suitability for educational purposes and the benchmarking of security analysis tools.

This thesis is structured as follows. Chapter 1 introduces the subject of the thesis, the motivation behind it, its objectives, and the methods for achieving them. In Chapter 2, we discuss the findings of both the literature review and the analysis of the state of the art in existing OWASP MASTG reference apps. Chapter 3 provides essential background knowledge on the OWASP MAS ecosystem as well as on key Android concepts discussed and used in this thesis. Chapter 4 discusses the design and project architecture for the to-be-developed reference apps and includes design decisions regarding vulnerability selection. In Chapter 5, each implemented vulnerability is covered in detail, supplemented with code snippets, implementation decisions, and challenges faced during their development. Chapter 6 concerns itself with the results of our work. Our developed apps are compared with the existing MASTG reference apps catalog, and their contributions are assessed. Chapter 7 discusses possible threats to the validity of our work, as well as any problems left open or newly discovered. In Chapter 8, we look forward to future work and outline how this thesis could be extended upon. Chapter 9 then summarizes our work and closes out the thesis.

2. Related Work

In this chapter we discuss two types of related work. The first consists of existing OWASP MASTG reference apps for Android. The second is about academic publications that rely on the MASTG.

2.1. State of the Art in OWASP MASTG Reference Apps

There are 19 MASTG reference apps listed on the OWASP page [2] and its corresponding GitHub repository [3] at the time of writing. To evaluate the state of the art of MASTG reference apps, we examine each one of them in order to identify and categorize their implemented vulnerabilities. Only eight of these apps have both available source code and some form of documentation of their vulnerabilities. The vulnerabilities were then mapped to their corresponding OWASP Mobile Application Security Weakness Enumeration (MASWE) counterparts, which lists and discusses the vulnerabilities that the MASTG tests for.

2.1.1. Methodology

We evaluated 19 MASTG reference apps as to whether their source code is available online, and if so, whether they have any documentation of their weaknesses. The app selection includes four "Android Uncrackable" reference apps, named L1 [4], L2 [5], L3 [6] and L4 [7]. None of them have any documentation on their implemented weaknesses and are excluded from this review because of it. The other apps excluded for the same reason are: Android License Validator [8], DVHMA [9], Digitalbank [10], DodoVulnerable-Bank [11], disable-flutter-tls-verification [12], and MASTestApp-Android-NETWORK [13]. The MASTG-Hacking-Playground-Kotlin-App [14] has been excluded from this literature review because we have included and evaluated an identical Java version of this app.

The following apps have been included in the review: AndroGoat [15], DIVA Android [16], InsecureBankv2 [17], MASTG-Hacking-Playground [18], OVAA [19], InsecureShop

[20], Finstergram [21], and Android BugBazaar [22]. A summary of this selection process can be seen in Table 2.1.

2.1.2. Findings

Across the eight evaluated OWASP MASTG reference apps, 163 weaknesses were implemented. We mapped them to their respective MASWE counterparts and grouped them by their vulnerability category (Storage, Crypto, Auth, etc.). An overview of the apps and their implemented vulnerabilities can be seen in Table 2.2.

Lack of Vulnerability Documentation

A recurring issue with many of the reference apps lies in their documentation of the implemented vulnerabilities. 11 of the reference apps listed by the OWASP MASTG do not have any documentation of their weaknesses at all, which led to their exclusion from this evaluation. The apps that do include some form of documentation of their vulnerabilities do so in a very minimal way. Most often, the documentation consists of a single list that merely names the implemented weaknesses. None of the apps map the implemented weaknesses to their OWASP MASWE counterparts. Only two apps [18, 21] actually document what the weaknesses entail, how they are implemented, and how to fix or exploit them.

This additional documentation is not necessary for the apps to function as a penetration testing practice environment. But lacking it does reduce their value as an educational resource in reference to the MASTG. A simple mapping of the implemented vulnerabilities to their respective MASWE counterparts helps mobile developers better connect the theoretical concepts provided by the MAS project with the practical environment of the reference apps. By providing no documentation on the implemented weaknesses, developers using the apps to learn are unable to verify which vulnerabilities they have found and which ones they have not. This hinders them in assessing their skills in detecting and exploiting mobile security vulnerabilities, which negatively impacts the educational value provided by the apps. The apps adequacy and function as a mobile security tool testing benchmark are much affected in the same way. If there is no documentation of the vulnerabilities implemented, it becomes extremely difficult to reliably assert how many of the vulnerabilities found by the tools are part of the intentionally created set and how many of the intentionally implemented ones have been missed by the tools. This means that without any documentation of the implemented vulnerabilities, the MASTG reference apps are just as unfit for benchmarking SAST

Table 2.1.: Overview of the availability of source code and documentation of MASTG reference apps, and whether they were included in our analysis

OWASP MASTG Reference Apps			
App name	Source code	Documentation	Included
Uncrackable L1			
Uncrackable L2			
Uncrackable L3			
Uncrackable L4			
License Validator			
DVHMA			
Digitalbank	✓		
Dodo Vulnerable Bank	✓		
disable-flutter-tls-verification	✓		
MASTestApp-Android-NETWORK	✓		
MASTG-Hacking-Playground-Kotlin-App	✓	✓	
AndroGoat	✓	✓	✓
DIVA Android	✓	✓	✓
InsecureBankv2	✓	✓	✓
MASTG-Hacking-Playground	✓	✓	✓
OVAA	✓	✓	✓
InsecureShop	✓	✓	✓
Finstergram	✓	✓	✓
BugBazaar	✓	✓	✓

Table 2.2.: Mapping of MASTG reference apps vulnerabilities to their MASVS categories

OWASP MASTG Reference Apps									
App name	# Vuln.	Stora.	Crypt.	Auth	Netwo.	Platf.	Code	Resil.	Priva.
AndroGoat	24	7	1	1	4	8	1	2	
DIVA	13	5		5			3		
InsecureB.	25	4	2	4	1	8	1	5	
Playground	15	5	1		1	3	3	2	
OVAA	18	1	1			10	6		
Ins.Shop	19	2		2	1	13		1	
Finster.	5					3	2		
BugBaza.	44	7		1		23	10	3	

and DAST tools as any real-world apps dataset. The apps we develop aim to address this issue by documenting all implemented weaknesses, including a mapping of the implemented vulnerabilities to their respective MASWE counterparts. Additionally, each implemented vulnerability will contain documentation on where it is found, which lines of code are relevant, how it can be exploited, and sometimes how it can be fixed.

Uneven Distribution of Vulnerability Coverage

At the time of writing, there are a total of 117 MASWE vulnerabilities across eight categories documented by OWASP [23]. Of these 117 vulnerabilities, only 42 are actually covered by any of these reference apps.

The eight apps together have 163 implemented weaknesses that can be mapped to only 42 different types of vulnerabilities as defined by MASWE. Because there are 117 different vulnerability types, this leaves a total of 75 vulnerabilities, as documented and classified by OWASP MASWE and referenced in the MASTG, completely without representation. This means that the vulnerabilities implemented in the MASTG apps serve as a reference to only a minority of the MASWE vulnerabilities. While the vulnerability types that make up this minority are covered on average almost four times, the vast majority of vulnerability types goes completely without a practical reference in the MASTG apps.

To make this point more clear; the ten most frequently covered vulnerabilities are implemented a total of 87 times, as seen in Figure 2.1. This means that they represent 53.4% (87/163) of all implemented vulnerabilities, despite only covering 8.5% (10/117) of all vulnerability types as referenced by MASTG. The most frequently implemented vulnerability is MASWE-0066, which deals with insecure intents. Of the ten most frequently covered vulnerabilities, five are of the category MASVS-Platform.

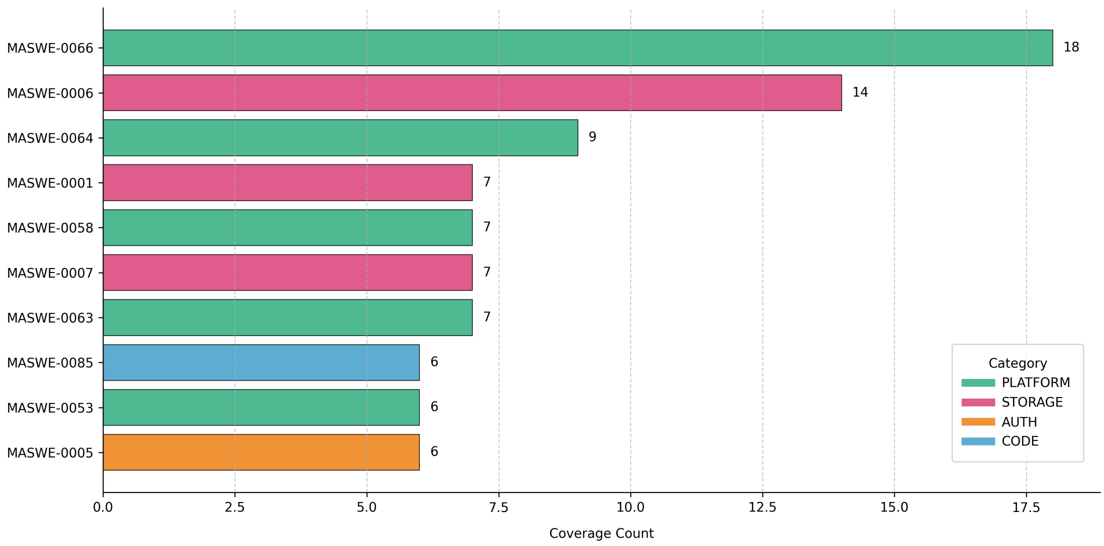


Figure 2.1.: Ten most frequently covered types of OWASP MASWE vulnerabilities

Coverage of Vulnerability Categories

There is not only a favoring of certain specific MASWE vulnerabilities, but an uneven coverage of the vulnerability categories themselves as well. The percentage-wise coverage of MASVS categories can be seen in Figure 2.2. Some categories see a high percentage of their vulnerabilities covered, such as MASVS-PLATFORM with 72.7% or MASVS-STORAGE with 66.7%. Others see very little to no coverage at all, such as MASVS-CRYPTO with 21.1% and MASVS-AUTH with only 10%. Worst of all is MASVS-PRIVACY, with a vulnerability coverage of 0%.

Viewing the coverage in absolute rather than percentage values makes it look slightly more even. As seen in Figure 2.3, all but two categories have between two and seven of their types of vulnerabilities covered. The only outliers are the categories MASVS-PLATFORM, which has a total of 16 out of 22 vulnerabilities implemented, and MASVS-PRIVACY, which has no vulnerabilities covered at all. This shows that vulnerabilities are covered rather evenly across the 8 categories, regardless of the number of vulnerabilities each category has to offer, which explains the large differences in percentage-wise coverage seen in the previous Figure 2.2. This does not mean that the coverage of the different MASVS categories is proportional, as larger categories should have a larger group of vulnerabilities implemented than smaller ones when viewed in absolute numbers. This further reinforces the fact that the vulnerability categories are not covered evenly.

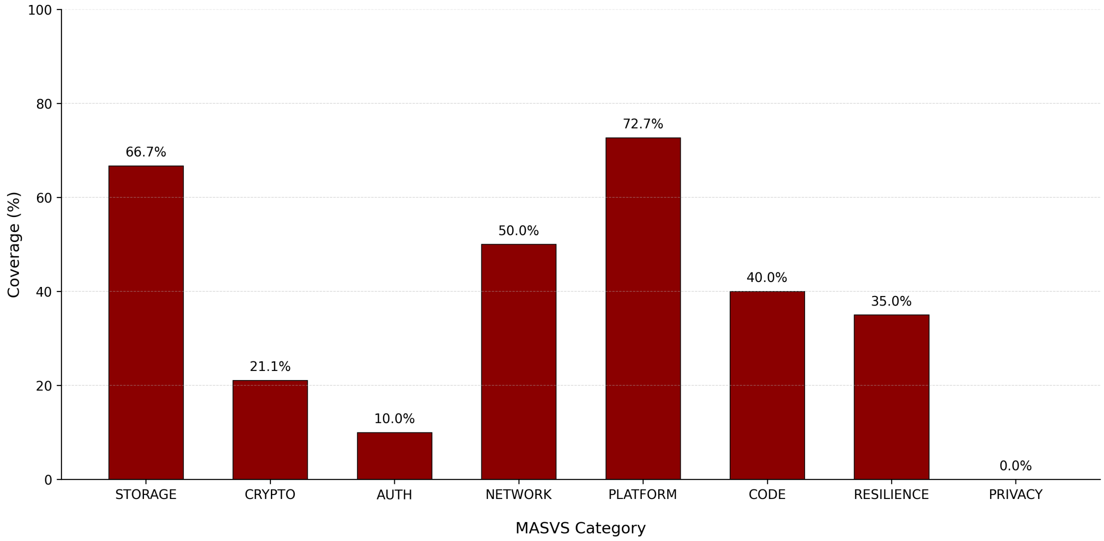


Figure 2.2.: Relative coverage of MASVS categories

The remarkably high coverage of MASVS-PLATFORM vulnerabilities coincides with the findings shown in Figure 2.2; not only are 5 of the 10 most frequently covered vulnerability types part of MASVS-PLATFORM, but the category itself is also the most broadly implemented category when compared to the others. The complete lack of coverage of MASVS-Privacy vulnerabilities can in part be accounted for by the fact that the category itself is newer than the other seven, having been introduced in MASVS version 2.1.0 [24].

Nevertheless, it can be concluded that the categories are unevenly covered, with MASVS-PLATFORM receiving a great deal more attention than the other categories, and MASVS-PRIVACY receiving none. The coverage of many categories in these MASTG reference apps is lacking, with only 21.1% and 10% of all MASVS-CRYPTO and MASVS-AUTH vulnerabilities being covered, respectively. This stands in stark contrast to the amount of material provided on them by the OWASP MAS project. They represent a large part of the mobile security that OWASP deals with, considering they make up a third (39/117) of all vulnerabilities documented by the MASWE and tested for by the MASTG.

The reference apps we develop aim to solve this issue by focusing on vulnerability types that have not been implemented in existing MASTG reference apps, as well as by shifting the focus from over-represented categories such as MASVS-PLATFORM to more neglected ones, such as MASVS-CRYPTO and MASVS-AUTH.

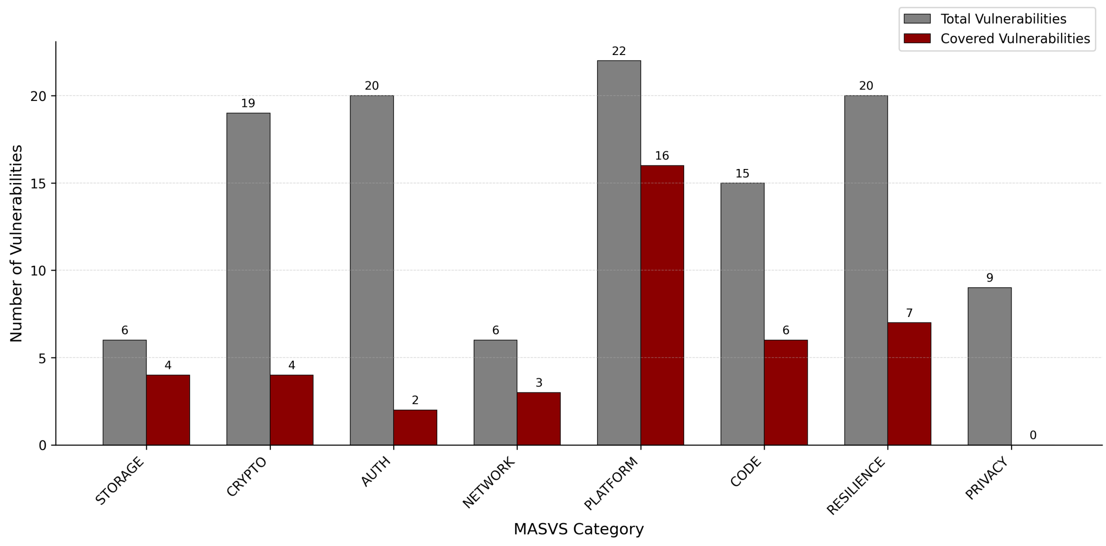


Figure 2.3.: Absolute coverage of MASVS categories

2.2. Literature Review: Academic Relevance of OWASP MASTG

The OWASP MASTG is the industry standard for mobile app security. But what relevance does it have in the academic world, and what exactly is it used for? To answer these questions, a literature review was conducted to evaluate academic publications on their usage of the OWASP MASTG, as well as other parts of the OWASP MAS project.

This literature review was conducted by searching for relevant papers that include and deal with the keywords "OWASP" and "MASTG" on Google Scholar. Only papers written in English were considered. Furthermore, all evaluated papers have been published in the last three years, with no paper having been published prior to 2023. The top 22 papers matching these criteria have been carefully read, with seven of them discarded because their research focus did not match our defined scope. This left 15 papers [1, 25–38], which were then evaluated for their usage of OWASP MASTG and grouped by similarity.

Of the 15 papers evaluated, eight [1, 25–28, 30, 33, 34] used OWASP MASTG to evaluate real-world Android apps based on whether they adhere to the security standard set by OWASP MASVS. Six papers [1, 25–28, 30] mapped Android app security vulnerabilities to the OWASP MASVS. Three papers [26, 27, 29] made use of the penetration testing guide, as laid out by OWASP MASTG, in order to conduct penetration tests on Android apps. Three papers [32, 35, 38] used the MASTG reference apps by evaluating vulnerability assessment tools on them in order to verify their accuracy in finding such vulnerabilities. Two of these papers [35, 38] then compared their findings to results found when conducting the same tests on real-world apps. Lastly, four papers [31, 36–38] mentioned the OWASP MASTG in discussion. This was done either by the authors, comparing the OWASP mobile project to other Mobile Security Frameworks, or by interviewees, as a tool used. An overview of the different papers and how they made use of the OWASP MAS project can be seen in Table 2.3.

Table 2.3.: Overview of how scientific papers make use of the OWASP MAS project

OWASP MAS usage by scientific papers					
Paper	Evaluating real-world apps	Vulnerability mapping	Penetration testing guide	Reference apps	In Discussion
1 [1]	✓	✓			
2 [25]	✓	✓			
3 [26]	✓	✓	✓		
4 [27]	✓	✓	✓		
5 [28]	✓	✓			
6 [29]			✓		
7 [30]	✓	✓			
8 [31]					✓
9 [32]				✓	
10 [33]	✓				
11 [34]	✓				
12 [35]				✓	
13 [36]					✓
14 [37]					✓
15 [38]				✓	✓

Cryptographic Security Shortcomings in African Mobile Financial Applications

Chiboora et al. [30] used the OWASP MASTG and the MASVS to evaluate the mobile security of 18 Android apps belonging to different African financial institutions. They chose MASVS as a testing framework because of its focus on being a guide and checklist for evaluating the security of mobile applications. They applied the strictest level of MASVS, L2, and focused solely on unauthenticated client-side testing, following the black-box testing approach. They did not implement all MASVS tests. Nevertheless, their findings in terms of security shortcomings are quite damning. All 18 apps allowed the taking of screenshots of sensitive data. 14 of the apps used outdated and thus insecure cryptographic protocols such as MD5, SHA-1, and DES. 10 apps used security libraries that are outdated and known to have vulnerabilities. Furthermore, eight apps were using insecure random number generation for their cryptographic protocols, and five apps had no certificate pinning. This shows a clear and dramatic shortcoming in terms of cryptographic mobile security. All of the shortcomings mentioned above are well documented by the OWASP MASWE, tested for by the MASTG, and are part of the security controls of the MASVS. This shows that the OWASP MAS project has a great use case in the real world, and is far from being outdated or redundant. In a survey conducted by the authors of this study, 78% of respondents said they avoided implementing security measures due to a lack of expertise and knowledge about mobile security. We aim to increase the coverage of MASVS-CRYPTO vulnerabilities through our intentionally vulnerable reference apps. This addresses an important shortcoming in the current landscape of available MASTG reference apps, which do not cover the MASVS-CRYPTO category sufficiently. This decision concurs with the issues found and discussed in this paper by enabling mobile developers to deepen their expertise and knowledge of cryptographic vulnerabilities in a hands-on way.

Lack of Basis for Mutation Testing in Android Application Security

Vasconcelos et al. [32] explored Mutation Testing as an approach for Android app security testing. The authors used the OWASP Top 10 Mobile Risks to assess common Android security vulnerabilities and created mutation operators based on these vulnerabilities. They then performed security tests on the mutants generated by the mutation operators, in the hope that these mutants would be detected as vulnerabilities. They applied this procedure to 10 Android reference apps from the MASTG reference app catalog. They found that four of the eight generated mutant operators were not represented among the mutants at all, while two other were heavily overrepresented. To be more specific, all three tapjacking vulnerability mutant operators and the "Implicit-PendingIntent" operator have no representation in the apps, whereas "ImproperExport" and "HardcodedSecret" are both heavily overrepresented. This corroborates our own findings upon analyzing the implemented vulnerabilities in the reference apps. Only one of the 163 vulnerabilities implemented by the MASTG reference apps deals with tapjacking attacks. The same applies to implicit pending intents. On the contrary, eight vulnerabilities deal with hard-coded secrets. Because of uneven distribution of implemented vulnerabilities, some mutants are generated more frequently than others. This shows that the large number of MASWE vulnerability types that go without a practical implementation in the reference apps affects the academic world as well. By focusing on these vulnerabilities for the development of our reference apps, we aim to address this issue.

Security Shortcomings in the American mHealth Sector

Stevenson and Das [34] used the OWASP Mobile Audit to evaluate 95 real-world apps that are part of the mobile health sector in the USA. The OWASP Mobile Audit is an OWASP project that applies SAST aligned with the MASTG to detect mobile security vulnerabilities. The findings were quite devastating. These apps dealing with sensitive user health data had, on average, around 9,000 vulnerabilities, with two thirds of them being of medium to critical severity. Around half of the evaluated apps transmit data containing personal health information with no encryption over HTTP, or make use of heavily outdated cryptographic algorithms. Furthermore, they found 2,252 instances of unnecessarily exported broadcast receivers and 1,232 cases of exported permissions without defined protection levels.

This paper underlines the significant shortcomings of apps in the cryptographic sector of mobile security, even when it comes to sensitive user data. All of the found vulnerabilities are well documented by the MASWE, and would not exist anymore if developers followed the standards laid out by the OWASP MAS project. The authors conclude that the overall lack of security measures implemented is further worsened by the fact that the developers follow insecure coding practices. The authors specifically recommend the OWASP MASVS as the security standard that mobile developers should adopt. This paper emphasizes the real-world importance of the OWASP MASVS, the MASTG, and the related reference apps. Many of the vulnerabilities detected by the OWASP Mobile Audit deal with cryptographic issues, which is a MASVS category with very limited coverage by existing MASTG reference apps. The reference apps we develop aim to address this issue by focusing on vulnerabilities documented in MASVS-CRYPTO.

Security Assessment for Digital Wallet Applications

Saifulhakim and Fajar [25] made use of the OWASP MASTG and MASVS to assess the security standards of a digital wallet payment partner mobile app. The authors applied both dynamic (DAST) and static (SAST) application security testing. They mapped the found vulnerabilities to the MASWE and used the OWASP Risk Rating Methodology to assess the security risks of the detected vulnerabilities. They identified vulnerabilities using both the static and dynamic approach. The paper found that of the 84 MASVS verification points, 68 were met, 10 were deemed not applicable, and six failed due to vulnerabilities found. The authors referenced multiple papers and articles demonstrating that the OWASP MAS project presents a viable solution for security assessment. They note that financial transaction apps are particularly common to have a high vulnerability rate, and that insecure data storage occurs more frequently than other vulnerabilities types.

This paper shows that MASTG reference apps should include both dynamic and static vulnerabilities if they desire to mimic real-world app vulnerabilities. It goes on to discuss the high frequency in which storage vulnerabilities occur. The reference apps we develop build on these findings by including both static and dynamic vulnerabilities. Our reference apps aim to implement MASWE vulnerabilities related to storage. By doing so, our apps reflect vulnerabilities as they are found in real-world apps.

Hardening Techniques in Mobile Applications

The OWASP MASVS advises mobile app developers to implement so called hardening techniques. They serve as a layer of device protection by preventing tampering with the device and detecting jailbreaks. Steinböck et al. [38] investigated the extent to which real-world apps adhere to the MASVS and implement such hardening techniques. To do so, they created HALY, a framework that allows for static as well as dynamic analysis of the adoption of hardening techniques in mobile apps. They discuss how MASVS-RESILIENCE specifies multiple hardening techniques that apps should implement as to conform to modern security standards. They used the MASVS as a base of operations since the MASVS is the standard used for Android apps that opt into Google Play's independent security review process [38]. This shows the importance of the recommendations and practices laid out in the standards established by the OWASP MAS project. Their framework HALY is built to detect vulnerabilities in three of the four categories defined by MASVS-RESILIENCE. The authors validated the accuracy of their framework by testing it on seven open source apps, including four Android and two iOS OWASP reference apps. They proceeded to use their framework to evaluate 2,646 apps, both available on Android and iOS, for their use of hardening techniques. Their findings show that iOS apps heavily under-perform when it comes to implementing hardening techniques, as opposed to apps on Android devices. Of the 2,646 apps evaluated, about 75% of Android and 25% of iOS apps implement more than half of the protection techniques recommended by MASVS-RESILIENCE. Only 26 Android apps adopt all eight, while only one iOS app adopts all seven. This paper shows the real-world importance of the MASVS through its role as the standard used in Google Play's data safety section. It displays the merit of the MASTG reference apps and the MASVS to the academic world as a basis on which to conduct important research. Their findings show that there are still many mobile apps that do not follow the security best practices defined by OWASP. The MASTG reference apps we develop provide a basis on which future academic work similar to this paper can build on. They furthermore serve as an example from which mobile developers can learn to improve the security standards of mobile apps used globally.

3. Background

This chapter briefly covers the most important Android concepts and components that are relevant to this thesis. Cryptographic algorithms and related topics are deliberately excluded, as they lie outside the scope of our work.

3.1. Android Internals

Core Android Components

Android `Activities` [39] are the entry point for an Android app's interaction with the user. Android apps are also launched from an activity with special privileges that is called a launcher activity. An activity provides the screen in which the app displays its User Interface (UI). An app needs to have an activity for each screen it wants to display. Activities call each other in order to perform the different actions that the app provides. For an app with login functionality, this might include a main activity which is the starting point of the app, a registration activity, a login activity and a profile activity.

An Android `Service` [40] is an Android application component designed for long running background processes. In contrast to activities, services do not provide any form of UI. When a service runs in the foreground, it means that they execute a process noticeable to the user, such as playing an audio. Otherwise they run in the background where the user won't notice them, for example when dealing with app storage management.

`Content providers` [41] are the standard Android interface used by an app to access data stored by itself or other apps on the device. They can be used to grant other apps permissions to certain files while keeping others inaccessible. Content providers encapsulate data and allow the app to specify which app has which permissions on which files and directories. This way, other apps cannot directly access and meddle with data stored internally within an app, and have to use the proper and app-defined way of requesting file permissions.

The `AndroidManifest.xml` [42] is the heart of every Android mobile application, and every app is required to have one. It has many functions, one of which is declaring all components that are part of the app. This includes all activities, services, and content providers. It defines the permissions other apps need to access the components of the app. The Android manifest goes on to define the app's settings on concepts such as backup-creation and whether the app is debuggable or not.

Inter-Component Communication (ICC)

`Intents` [43] are the standard way for the components of an Android app to communicate with one another. Activities make use of intents to call one another, as well as for starting services.

`URIs` [44], short for Uniform Resource Identifiers, are strings in Android used to identify resources such as images, files, or data shared by content providers. They share similarities with common file paths but can be used to reference a wider range of different resources. The Android system enforces URI permissions that control who can read which resources. When a content provider gives an entity permission to read a certain file, it does so by returning the designated `content URI`.

Data Storage

`SharedPreferences` [45] is an Android specific API that allows apps to store data persistently across user sessions. It is designed for storing small amounts of data in a dictionary-like key-value format. The data placed in `SharedPreferences` is stored in the app's private directory on internal storage.

`KeyStores` are a system designed for safely storing cryptographic keys, making them hard to extract but easy to use. Once a key is stored within a keystore, it can be used for cryptographic operations, but is hard if not impossible to extract. Modern keystores such as the `Android Keystore system` [46] allow apps to clearly define what the key can and cannot do, as well as under what circumstances it can be used and what user authentication is required to do so. It is the modern and secure Android way of storing and managing cryptographic keys.

`Application Sandbox` [47] is an Android security mechanism that isolates all apps from the system and from other apps. Through this, each app has its own dedicated storage area and runs in its own process. It prevents malicious apps from directly reading or modifying data from other apps. This mechanism is enforced at the operating system level. To access the storage of another app's sandbox or system components such as the camera or location, an app has to request the appropriate permissions. This design forces apps to communicate with one another through controlled channels, such as `Intents` or `Content providers`.

3.2. OWASP MAS

The OWASP Mobile Application Security (MAS) project [48] aims to represent the industry standard for mobile application security. It comprises three main components. The OWASP Mobile Application Security Verification Standard (MASVS) [49] defines the security measures a mobile app should have. It provides mobile developers with a security standard to adhere to for security best practices. The OWASP Mobile Application Security Weakness Enumeration (MASWE) [23] lists and discusses a wide range of mobile security vulnerabilities. This includes knowledge on how they can be introduced and how they can be avoided. The OWASP Mobile Application Security Testing Guide (MASTG) [50] is an industry standard manual for testing and evaluating the security of mobile apps. The MASVS lays out the requirements a mobile app has to fulfill for industry-standard security practices, the MASTG defines how these requirements can be tested, and the MASWE describes the vulnerabilities found through this testing process.

Part of the MASTG is a dataset of intentionally vulnerable applications that are hosted by OWASP MAS [2]. These apps serve as educational reference material for the MASTG.

4. Design

In terms of designing our apps, here we cover the system architecture, the frameworks, and the technologies used. Furthermore, the selection process of the implemented vulnerabilities is explained. We cover the rationale behind our decisions and the reasons why other approaches were considered but ultimately not chosen.

4.1. System Architecture

4.1.1. System Architecture Overview

A modular, multi-app system architecture is used for the overarching structure of this Android project. The implemented MASWE vulnerabilities are grouped by their respective MASVS categories. This means that all implemented vulnerabilities of the category MASVS-STORAGE are part of the same module `masvs_storage`, whereas the vulnerabilities from the MASVS-CRYPTO category make up their own `masvs_crypto` module.

Each module is its own Gradle module, with its own `AndroidManifest.xml`, activities, services, backup rules, and launcher activity. The advantage of this is a clean and structured separation of the different implementations by category. With our design, the categories are each their own Android app, and can be used and tested independently of one another. Another advantage is that vulnerabilities affecting the `AndroidManifest.xml` are contained within their category instead of impacting all vulnerabilities. Without this, a MASVS-PLATFORM vulnerability that sets the app to debuggable in the `AndroidManifest.xml` would result in all vulnerabilities having to include this debuggable vulnerability.

We use a shared-library approach to minimize the amount of redundant code. This means that all independent MASVS app modules make use of the same `common` Android library module, which has a `res` folder containing the resource values shared by all apps. This includes values regarding spacing, strings, font sizes, and color values. The `common` library module also includes layouts shared by all apps, and abstract activity templates, which are discussed in Subsection 4.1.3. A high-level diagram of this system architecture is shown in Figure 4.1.

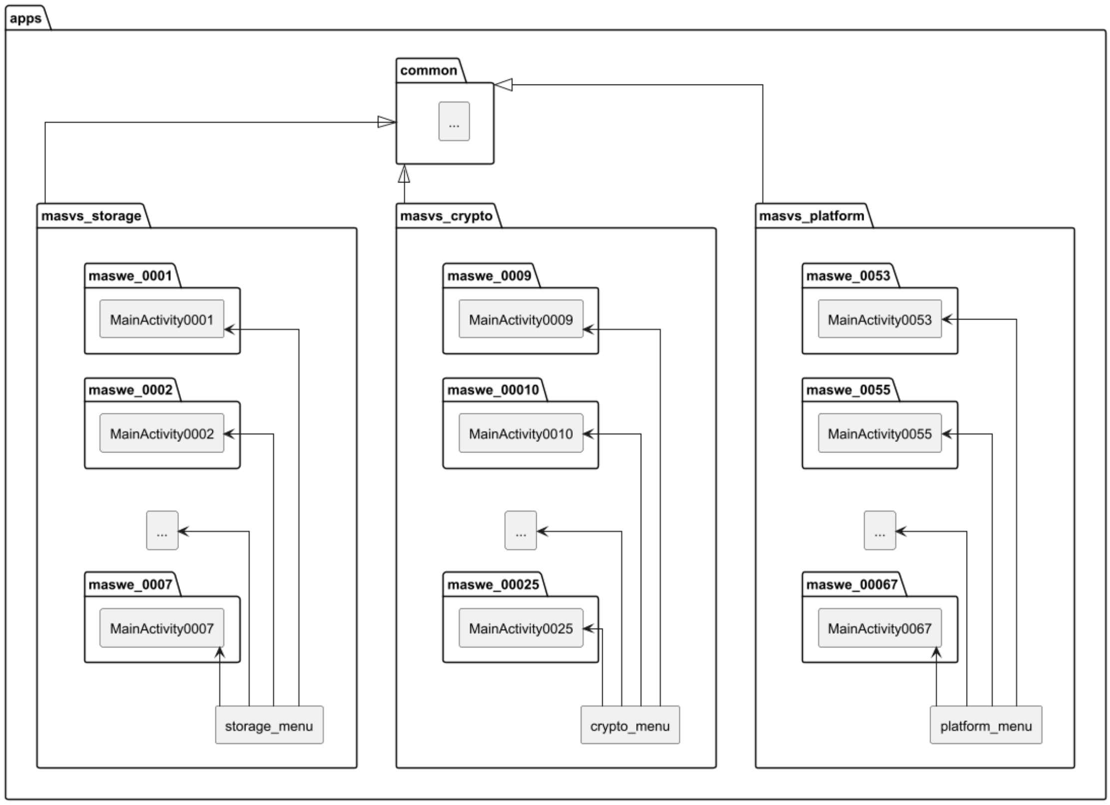


Figure 4.1.: High-level view of the Android apps system architecture

4.1.2. Feature-Oriented Structure Based on OWASP MASVS

Each app module is structured as follows. The launcher activity functions as the overview and hub from which all implemented vulnerabilities of the respective category can be selected. The vulnerabilities themselves are grouped into separate packages. These packages include all activities, services, and other files that are part of the vulnerability's code. Every vulnerability has a main activity that serves as the vulnerability's starting point. Furthermore, every package has a `RegisterActivity.java`, `LoginActivity.java`, and `ProfileActivity.java` that are used for user registration and login. This architecture is extended with other files depending on the vulnerability and its implementation, such as `EncryptionHandler.java`, which deals with cryptographic services. A high-level model of this structure can be seen in Figure 4.2.

This architecture further reinforces the clean separation and structure of the Android project. We package and isolate the code for each vulnerability's implementation and group the vulnerabilities themselves by their respective categories. This creates a clear and intuitive overview of which vulnerabilities have been implemented and where their code can be found. It allows for the isolated implementation, analysis, and testing of each vulnerability. This benefits both the development process and the usability of the apps.

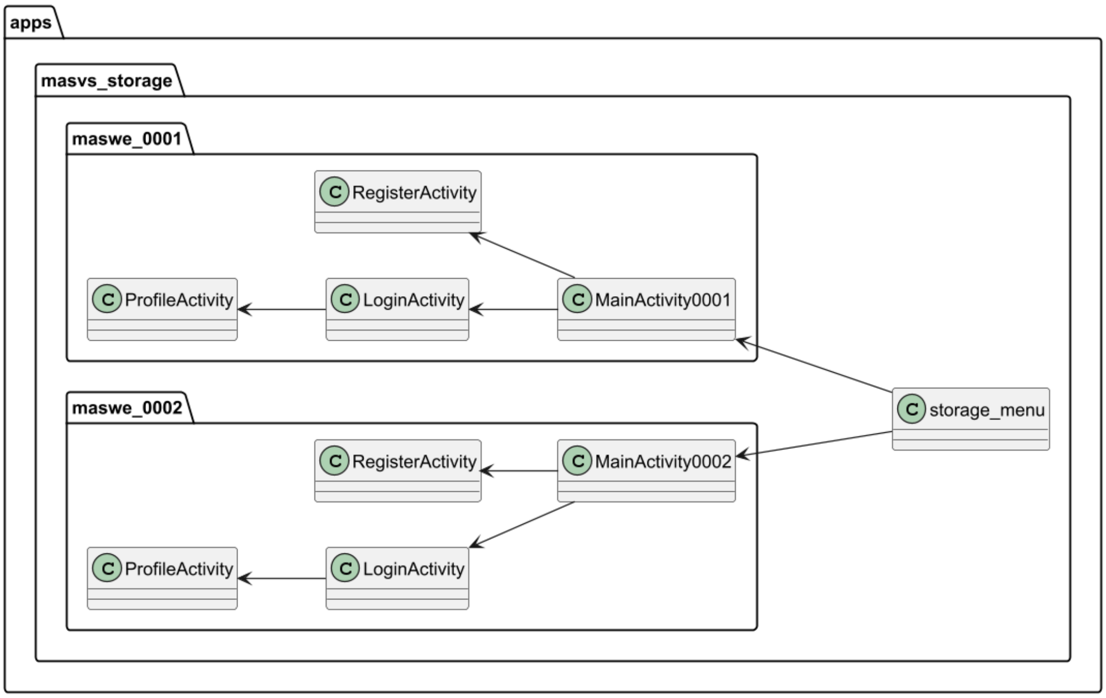


Figure 4.2.: High-level model of the feature oriented project structure

4.1.3. Template-Based Activity Architecture

We make use of custom templates to significantly reduce the amount of boilerplate code. All of our vulnerability implementations require a starting activity, as well as a registration, login, and profile activity. Between the different vulnerabilities, these might differ in code logic but are almost all identical in layout. Because of this, the `common` library module includes layout templates for all four of them. These are fitted with uniform styling, as well as the basic navigation buttons and user input fields expected of them.

In terms of code logic, the activities themselves share a lot of similarities and thus share boilerplate code as well. For example, many of the MASVS-CRYPTO vulnerabilities differ in the concrete implementation of cryptographic operations but are otherwise identical to each other, as well as to vulnerabilities of other categories. To make use of these shared pieces of logic and remove redundant code, the `common` module includes three abstract activity templates that all activities make use of. The first is `BaseActivityTemplate.java`, which all activities use, and the other two templates build on. The other two activity templates used are `BaseRegisterActivity.java` and `BaseLoginActivity.java`. The way this inheritance hierarchy works is shown in the high-level model seen in Figure 4.3.

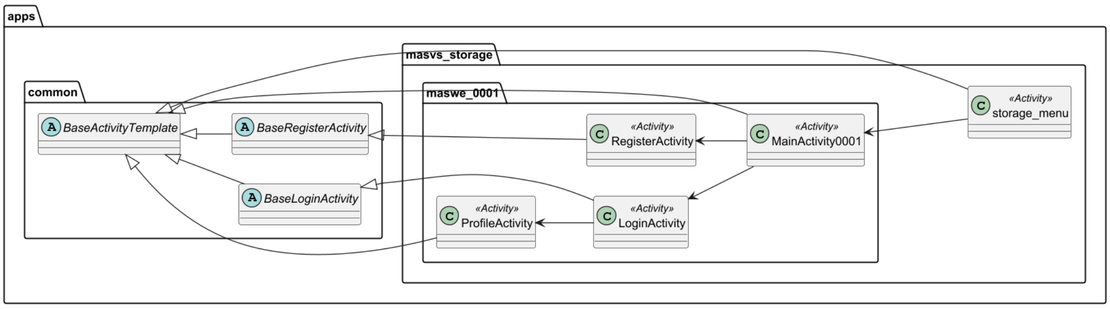


Figure 4.3.: High-level model of the inheritance templating approach used for the project

Base Activity Template

This template removes redundant code by implementing the functionalities shared by all activities in our Android project. A flow diagram of it can be seen in Figure 4.4. It provides the hook method `getScreenTitle()` that can be used by inheriting activity subclasses to set the screen title of the activity. It also provides methods for adding navigation buttons and mapping them to the activity classes that they link to.

Base Register Activity

`BaseRegisterActivity.java` builds upon `BaseActivityTemplate.java` and extends it with the registration code logic shared by all vulnerability implementations. This logic consists of handling and sanitizing user inputs, optionally encrypting the user password, and saving the newly registered user credentials in JSON format in a designated `SharedPreferences` file. A flow diagram of this template can be seen in Figure 4.5.

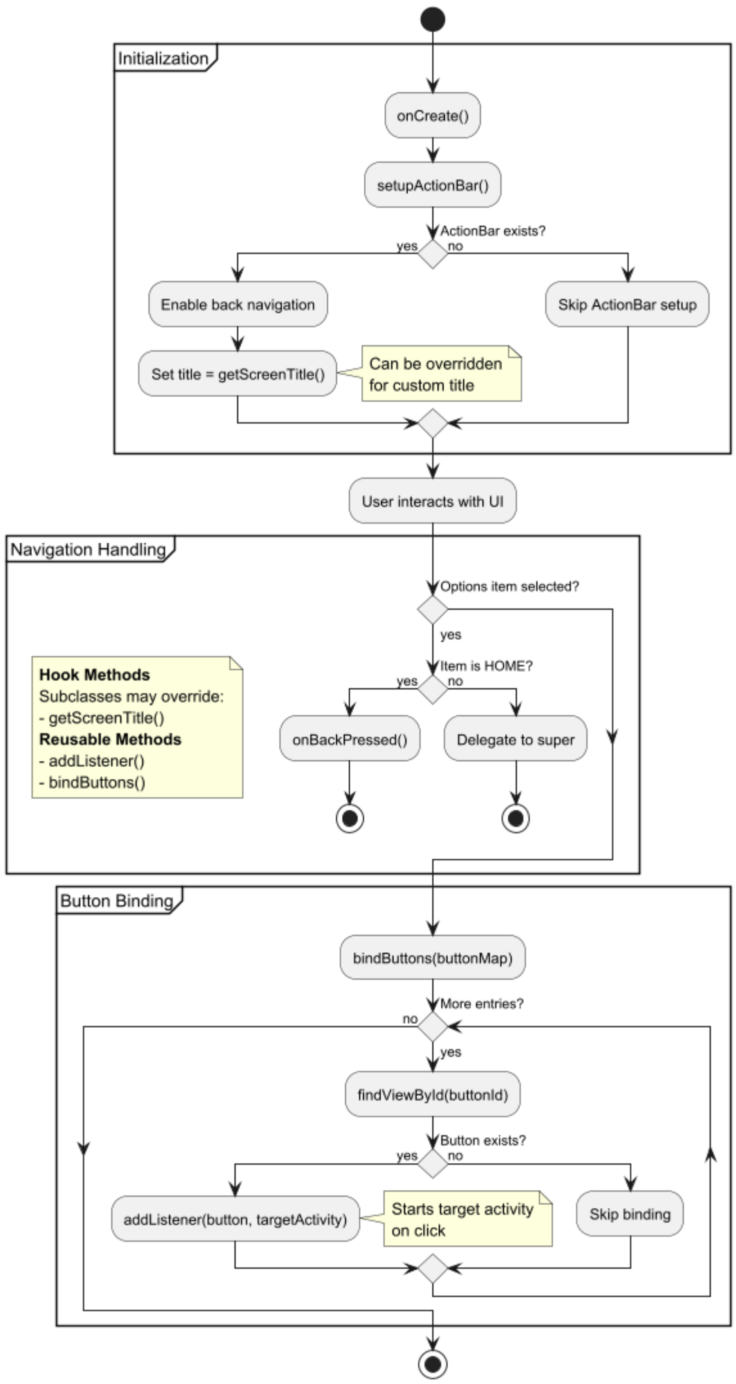


Figure 4.4.: Flow diagram of the base activity template

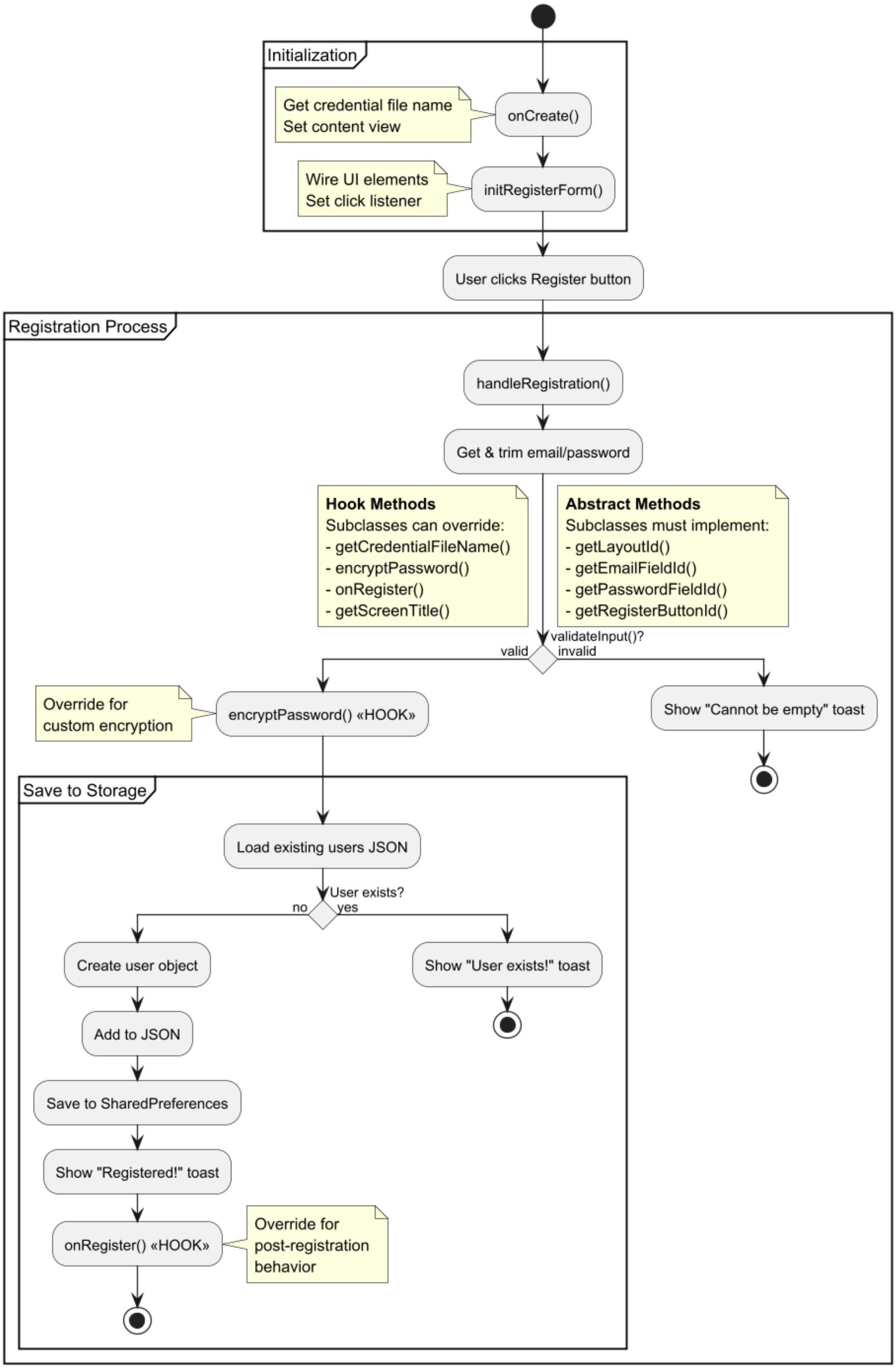


Figure 4.5.: Flow diagram of the register activity template

Any inheriting registration activity only needs to implement the abstract getters for the concrete layout ID and UI buttons. All other code is handled by the abstract class. The advantage of this is clear, as it greatly reduces the amount of boilerplate code. The template provides two different hooks. The first hook method, `encryptPassword()`, allows the inheriting registration activity to add custom encryption to the user credentials before storing them. This hook is where different MASVS-CRYPTO vulnerabilities introduce their vulnerable encryption methods. The second hook method, `onRegister()`, allows for custom post-registration behavior and logic.

Base Login Activity

`BaseLoginActivity.java` builds upon `BaseActivityTemplate.java` in a similar fashion to `BaseRegisterActivity.java`, as shown in Figure 4.6. It requires the inheriting activity class to set the same getters as the registration activity template, which are specific to each login activity implementation. The template sets up the activity and its UI elements, sanitizes user input, and loads stored user credentials. It proceeds to optionally decrypt the stored credentials before comparing them to the user input. The user is then directed to the specified target activity if the login attempt is successful.

It provides five different hook methods. `getCredentialFileName()` allows the inheriting login activity classes to specify the path to the file containing the user credentials, and hooking into `decryptPassword()` enables login activities to add custom decryption for the stored credentials. The subclasses can furthermore use the `verifyPassword()` hook method to customize how the password verification process looks like and use the `onLoginFailure()` hook to specify what the activity should do upon a failed login attempt.

4.2. Frameworks and Technologies

The apps are developed using native Android development in Java 11 and targeting API level 35, which translates to Android 15. The reason behind aiming for such a high API is that many of the existing MASTG reference apps suffer from being outdated, having not received any updates or maintenance in years. Because the OWASP MAS project is constantly evolving alongside the Android ecosystem, it would not make much sense to develop apps for an older Android version than the one the standard is based on, since the apps are supposed to be a reference resource for the standard. This ensures that the apps do not include vulnerabilities and technologies that are no longer relevant

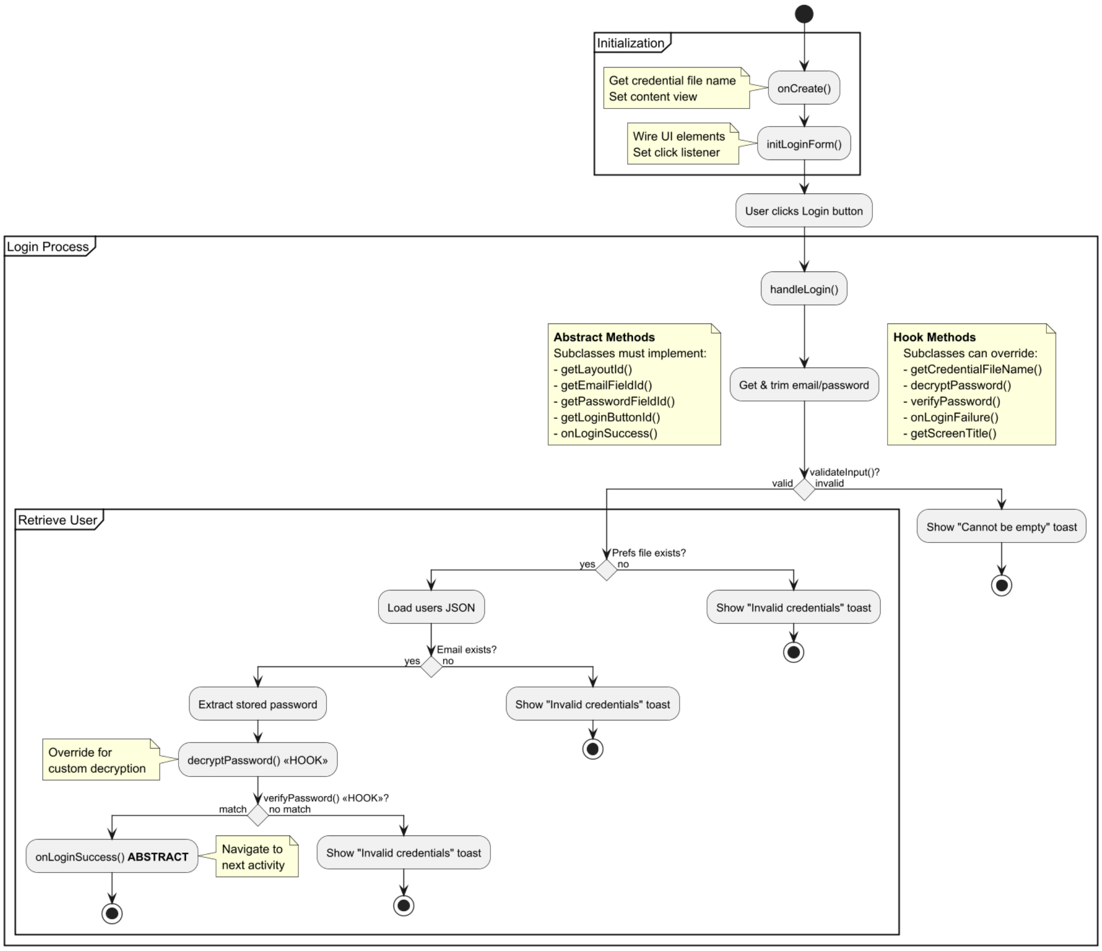


Figure 4.6.: Flow diagram of the login activity template

or have been removed in modern Android (and consequently in modern OWASP MAS) versions. The project uses Gradle with Kotlin DSL as its build automation tool. The core frameworks include the Android Framework, various APIs for cryptography, and AndroidX for backward compatibility.

Android Framework

We use Activities and Services for UI logic and background processing. By default, user credentials are stored within `SharedPreferences` files. This is a simple and native Android way to persistently store data. Intents are used for communication between our different components, such as Activities and Services. The Android Keystore provides a modern, safe, and simple way to store and use cryptographic keys.

APIs for Cryptography

For cryptographic algorithms and operations, we use both Java Cryptography Architecture (JCA) and Android-specific cryptography APIs. JCA is used as a less secure alternative to Android-specific APIs, which is useful for designing security vulnerabilities. Besides operations such as key generation, encryption, and signature, JCA is also useful for implementing cryptographic features deemed deprecated and insecure by Android-specific APIs, such as older KeyStores. The Android-specific APIs include, for example, the Android Keystore System (`android.security.keystore.*`) for a hardware-backed, native Android approach of managing cryptographic keys, and the Base64 API (`android.util.Base64`) for encoding keys.

AndroidX Libraries

The project makes use of AndroidX libraries to ensure the backward compatibility of our apps with older Android versions. This is especially relevant since the apps are based on Android 15, a relatively new Android version. We use AppCompat (`androidx.appcompat`) for backward-compatible UI elements, and all Activities extend `AppCompatActivity` to ensure our apps behave consistently and as intended across different Android API levels, especially the older ones.

4.3. Vulnerability Selection Process

At the time of writing, there are 117 vulnerabilities documented by OWASP MASWE, with two of them classified as deprecated [23].

Of these 115 viable vulnerabilities, we decided to focus on those that have received little to no coverage in existing MASTG reference apps. Another approach would have been to add more variations of vulnerability types already covered by other apps. This can add value to the existing reference app dataset as well, but only if these variations differ sufficiently from existing implementations. By focusing on vulnerabilities with little to no previous coverage, we ensure that the vulnerabilities implemented in our apps are unique. This way, they add to both the educational value of the MASTG reference apps and the usefulness of the reference apps for benchmarking mobile application security testing tools.

We decided to work on one category at a time. It appeared sensible for both educational and tool testing reasons to focus on having a few categories covered completely. The alternative would have been to touch on many categories, but only doing so to a limited extent. It is important to keep in mind that we design the MASVS categories to be separate apps that can be run, analyzed, and tested as independent units. For this purpose, it seems useful to have a small number of apps that are substantial in size and content, instead of many apps that provide little value individually.

In terms of which categories to start with, we decided to first focus on MASVS-STORAGE. With only six storage vulnerabilities described by the MASWE, it has the smallest number of vulnerabilities to implement, making it a good starting place. Our Literature Review (Section 2.2) showed the prominence of storage vulnerabilities in real-world apps [25], which underscores the importance of this category. MASVS-STORAGE exclusively deals with Android-native features. This creates a useful environment for becoming more familiar with the Android ecosystem and its functionalities.

Afterwards, we shift our focus to the categories that best fit our system architecture and existing implementations, which have a large number of vulnerabilities with little to no prior coverage by other apps. Because of these criteria, we decided to implement MASVS-CRYPTO next, as it fits nicely into our template approach without needing many adjustments to the code of our project. The category has the second largest number of vulnerability types without implementations, with only 4 of 19 possible MASVS-CRYPTO vulnerabilities having received previous coverage. Furthermore, cryptographic vulnerabilities were discussed by multiple papers [30, 34] included in our Literature Review as points of concern and focus.

5. Implementation

This chapter describes the concrete technical implementations of the selected MASWE vulnerability types. For each implementation, only code excerpts and architectural ideas relevant to the vulnerability are discussed instead of showing and going over the complete source code. Code that is not relevant to the vulnerability is omitted and indicated as such. The complete source code is available on our GitHub repository [51]. The chapter is structured according to the MASVS categories, and the vulnerabilities are sorted in the same way as done by OWASP MASWE.

5.1. MASVS-STORAGE

5.1.1. MASWE-0001: Logging Sensitive Data

OWASP MASWE Name	Insertion of Sensitive Data into Logs
Key Topics	App Logs, System Logs
Key Components	`RegisterActivity.java`
Programming Languages	Java

Table 5.1.: Overview of our MASWE-0001 implementation

To implement this vulnerability, we made use of the `onRegister` hook to call two methods whenever a new user registers. The first method writes the user credentials to system logs, while the second writes them to app logs, as shown in Listing 5.1.

```java
@Override
protected void onRegister(String email, String password) {
    userDataToSystemLogs(email, password);
    userDataToAppLogs(email, password);
}
```

Listing 5.1: User registration hook invoking logging of sensitive credentials

For system logs, we make use of the command-line tool `logcat`, as shown in Listing 5.2.

```java
private void userDataToSystemLogs(String email, String password){
    Log.d(LOG_TAG, "New User registered");
    Log.d(LOG_TAG, "User E-Mail: "+ email);
    Log.d(LOG_TAG, "User Password: " + password);
}
```

Listing 5.2: Writing user credentials to system logs using `logcat`

Listing 5.3 shows the second method. It takes the user credentials and writes them to the app's internal file system, which is part of its app logs.

```java
private void userDataToAppLogs(String email, String password){
    try {
        File logFile = new File(getFilesDir(), CREDENTIALS_FILE_NAME + ".txt");
        FileWriter writer = new FileWriter(logFile, true);
        writer.append("Login - Username: ")
                .append(email)
                .append(", Password: ")
                .append(password)
                .append("\n");
        writer.close();
        Log.d(LOG_TAG, "Logged credentials to app logs");
    } catch (IOException e) {
        Log.e(LOG_TAG, "Error writing to log file: " + e.getMessage());
    }
}
```

Listing 5.3: Writing user credentials to application log files

5.1.2. MASWE-0002: Insufficiently Protected Local Storage Data

OWASP MASWE Name	Sensitive Data Stored With Insufficient Access Restrictions in Internal Locations
Key Topics	`FileProvider`, File Permissions
Key Components	`AndroidManifest.xml`, `RegisterActivity.java`
Programming Languages	Java, XML

Table 5.2.: Overview of our MASWE-0002 implementation

The challenge of implementing this vulnerability stemmed from the fact that global file permissions have not been allowed by Android since Android 7.0 (API 24) [52]. For example, attempting to set the permissions of any file to world-writable or readable, as shown in Listing 5.4, will cause built-in Android security checks to throw a `SecurityException`.

```
SharedPreferences sharedPreferences =
    getSharedPreferences("maswe_0002_user_credentials",
    Context.MODE_WORLD_READABLE | Context.MODE_WORLD_WRITABLE);
```

Listing 5.4: Attempting to set permissions of a file to world read and writable

To still be able to implement this vulnerability, we made use of a custom `FileProvider`. This `FileProvider` attempts to share access to the file containing user credentials with anyone, without checking their permissions to do so. The relevant code for this is shown in Listing 5.5.

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="com.dkronig.masvs_storage.CustomFileProvider"
    android:exported="false"
    android:grantUriPermissions="true"
    android:permission="" >
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/file_paths" />
</provider>
```

Listing 5.5: Custom `FileProvider` that grants `Uri` permissions without checking access permissions

This `FileProvider` is called every time a new user registers by hooking into the `onRegister` method. We first create a new `Uri` for accessing the file storing the user credentials. It then creates a new `Intent` for sharing, attaches the file in plaintext, and prompts the user to share it with any app of their choosing. The calling of the `FileProvider` is shown in Listing 5.6.

This approach is less dangerous than setting file permissions directly, as it makes the user aware of the issue by directly prompting them to share the file. A breach of sensitive user data can only occur if the user chooses to share the credentials with third parties. We decided to settle on this approach since it did not appear feasible to implement this vulnerability more directly on new Android versions.

```java
@Override
protected void onRegister(String email, String password) {
    Uri uri = FileProvider.getUriForFile(
            this,
            this.getPackageName() + ".CustomFileProvider",
            new File(this.getFilesDir(), "maswe_0002_user_credentials.txt"));

    Intent share = new Intent(Intent.ACTION_SEND);
    share.setType("text/plain");
    share.putExtra(Intent.EXTRA_STREAM, uri);
    share.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
    startActivity(Intent.createChooser(share, "Share file via"));
}
```

Listing 5.6: Prompting the user to share a plaintext file containing user credentials

5.1.3. MASWE-0003: Unencrypted Backup

OWASP MASWE Name	Backup Unencrypted
Key Topics	Backup Device Conditions and Backup Device Flags
Key Components	`backup_rules.xml`
Programming Languages	XML

Table 5.3.: Overview of our MASWE-0003 implementation

Android does not enforce any encryption of backups made by Android applications. Instead, it checks whether the backup is encrypted and only proceeds with the backup process if this is the case. To counter this and implement the vulnerability, we specifically set the flag shown in Listing 5.7 to false. This ensures that Android does not hinder the backup from happening, allowing us to back up sensitive user data in plaintext. Part of the backup is the entire `SharedPreferences` directory, which includes `maswe_0003_user_credentials.xml`. This file contains the login credentials of everyone who used the `RegisterActivity.java` of this vulnerability.

```xml
<cloud-backup disableIfNoEncryptionCapabilities="false">
    <include domain="sharedpref" path="."/>
</cloud-backup>
```

Listing 5.7: Backup rules for `maswe_0003`

We further ensured that client side encryption of the backup is not enforced by excluding the flag `<... requireFlags="clientSideEncryption"/>` for each include statement. This is done for both the cloud backup and device-to-device transfer.

5.1.4. MASWE-0004: Sensitive Data in Backup

OWASP MASWE Name	Sensitive Data Not Excluded From Backup
Key Topics	Cloud Backup Flags, Device-Transfer
Key Components	backup_rules.xml
Programming Languages	XML

Table 5.4.: Overview of our MASWE-0004 implementation

To ensure that user credentials are included in the backup, we explicitly include every SharedPreferences file in which the user credentials are stored by all our apps by default. This includes the file maswe_0004_user_credentials.xml, which includes the user credentials of this vulnerability. We do this for both the cloud backup, as shown in Listing 5.8, and for device-to-device transfer, as shown in Listing 5.9.

```xml
<cloud-backup disableIfNoEncryptionCapabilities="false">
	<include domain="sharedpref" path="."/>
</cloud-backup>
```

Listing 5.8: Cloud backup rules for maswe_0004

```xml
<device-transfer>
	<include domain="sharedpref" path="."/>
</device-transfer>
```

Listing 5.9: Device-to-device transfer rules for maswe_0004

5.1.5. MASWE-0006: Insufficiently Encrypted Private Storage Data

OWASP MASWE Name	Sensitive Data Stored Unencrypted in Private Storage Locations
Key Topics	Hard-coded Encryption Key, Broken Outdated Encryption (DES), Encryption Key Stored on Filesystem
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.5.: Overview of our MASWE-0006 implementation

For this vulnerability, we used a hard-coded key that is stored in the same file where it is used. This is used in combination with DES (Data Encryption Standard), which is considered a broken and outdated encryption algorithm. This is shown in Listing 5.10. We then use this weak encryption on user credentials upon registration and save this sensitive data in `SharedPreferences`, as shown in Listing 5.11.

```java
public class EncryptionHandler {
    private static final String ENCRYPTION_KEY = "EncryptK";
    private static final String ALGORITHM = "DES/ECB/PKCS5Padding";
    private static final String KEY_ALGORITHM = "DES";

    // ... omitted ...
}
```

Listing 5.10: Setting up hard-coded encryption key and DES for cryptographic operations

```java
public String encryptData(String plaintext) throws Exception {
    SecretKeySpec keySpec = createKeySpec();
    Cipher cipher = Cipher.getInstance(ALGORITHM);
    cipher.init(Cipher.ENCRYPT_MODE, keySpec);

    // ... omitted ...
}
```

Listing 5.11: Using DES and a hard-coded encryption key for weak encryption

5.1.6. MASWE-0007: Unencrypted Shared Storage Data

OWASP MASWE Name	Sensitive Data Stored Unencrypted in Shared Storage Requiring No User Interaction
Key Topics	`MediaStore`, Unencrypted User Data, External Storage
Key Components	`RegisterActivity.java`
Programming Languages	Java

Table 5.6.: Overview of our MASWE-0007 implementation

Writing to shared storage without user interaction has become increasingly difficult in newer Android versions. However, it is still possible to do so using `MediaStore`. `MediaStore` files are not part of the sandboxed app environment and are visible to all other apps on the device. If the reading permissions are not set properly, other apps are furthermore able to read the `MediaStore` files without restrictions.

To implement this vulnerability, we first create a new `MediaStore` entry to store future user credentials in, as shown in Listing 5.12. We then write user credentials to this entry whenever a new user registers.

```java
private void createMediaStoreEntry(){
    ContentValues values = new ContentValues();
    values.put(MediaStore.MediaColumns.DISPLAY_NAME, FILENAME);
    values.put(MediaStore.MediaColumns.MIME_TYPE, "text/plain");
    values.put(MediaStore.MediaColumns.RELATIVE_PATH,
        Environment.DIRECTORY_DOCUMENTS);

    fileUri = getContentResolver().insert(MediaStore.Files.
            getContentUri("external"), values);
}

private void writeToSharedStorage(String content) {
    try (OutputStream out = getContentResolver().openOutputStream(fileUri)) {
        assert out != null;
        out.write(content.getBytes());
        out.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Listing 5.12: Creating a `MediaStore` entry and writing user credentials to it

5.2. MASVS-CRYPTO

5.2.1. MASWE-0009: Insecure Cryptographic Key Generation

OWASP MASWE Name	Improper Cryptographic Key Generation
Key Topics	Insufficient Entropy, Insufficient Key length, Broken Encryption
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.7.: Overview of our MASWE-0009 implementation

We implemented this vulnerability in three different ways. Generating the cryptographic key used for encryption and decryption of user credentials is done using pseudo-random numbers with very little randomness. The cryptographic key generated is of insufficient length and is furthermore used with an encryption algorithm considered broken.

For generating our encryption key, we used a custom source of randomness. This is shown in Listing 5.13. For creating the source of randomness, we used a very predictable seed that is hard-coded into the `EncryptionHandler.java` file, which manages the entire encryption process. For generating pseudo-random numbers from this seed, we then use SHA1PRNG, which is extremely outdated.

```java
private static final String ALGORITHM_PRNG = "SHA1PRNG";
private static final String KEY_SEED = "01234567";
private static final int DES_KEY_SIZE = 56;

private static SecretKey generateKey() throws Exception {
    byte[] keySeed = KEY_SEED.getBytes(StandardCharsets.UTF_8);

    SecureRandom random = SecureRandom.getInstance(ALGORITHM_PRNG);
    random.setSeed(keySeed);

    KeyGenerator keyGenerator = KeyGenerator.getInstance(ALGORITHM_DES);
    keyGenerator.init(DES_KEY_SIZE, random);

    return keyGenerator.generateKey();
}
```

Listing 5.13: Generating a DES key of short length using a weak pseudo-random number source and a hard-coded seed

This predictable source of pseudo-random numbers is then used to generate our cryptographic key, as shown in the same Listing 5.13. For the cryptographic algorithm, we chose DES (Data Encryption Standard), which is an old and broken cryptographic algorithm. Broken here means that the encryption created by DES can be cracked by modern devices, rendering it obsolete. In addition to this, the DES key length is set to 56 bits. This, together with our insufficiently random source of pseudo-random numbers, poses a great threat to the functionality of the encryption used here, even if a modern cryptographic algorithm had been chosen in place of DES.

5.2.2. MASWE-0010: Insecure Cryptographic Key Derivation

OWASP MASWE Name	Improper Cryptographic Key Derivation
Key Topics	PBKDF2, AES, Insufficient Salting and Iterations
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.8.: Overview of our MASWE-0010 implementation

For this vulnerability, we decided to use Password-Based Key Derivation Function 2 (PBKDF2) to derive an Advanced Encryption Standard (AES) key, which is then used for encrypting and decrypting user credentials of the app. Both PBKDF2 and AES are considered secure if used properly. Since this vulnerability is about improper derivation of the cryptographic key, we focused on making the derivation process as exploitable and vulnerable as possible.

The idea behind PBKDF2 is to repeatedly hash a password into a strong cryptographic key. Adding a salt, which is a random and unique value added to the initial password, and repeating the hash process for a large number of iterations are required to make PBKDF2 cryptographically secure. For our PBKDF2 key derivation, we used a combination of vulnerable parameters, as shown in Listing 5.14. We use only 10 hashing iterations, whereas the recommended number of iterations for secure usage by OWASP since 2022 is 600,000 [53]. For the salt, we use a hard-coded, static 16 byte array of 0's, making the salting process useless. Combining this with the use of a weak, hard-coded password for the hashing process means that the PBKDF2 key derivation implemented here is completely insecure, making the AES key used for encrypting user credentials easily retrievable by any attacker.

```java
private static final int ENCRYPTION_ITERATIONS = 10;
private static final int KEY_LENGTH = 128;
private static final byte[] SALT = new byte[16];
private static SecretKey secretKey;
private static final String PASSWORD_FOR_KEY_DERIVATION = "password";

private static SecretKey generateKey() throws Exception {
    PBEKeySpec secretKeySpec = new PBEKeySpec(
        PASSWORD_FOR_KEY_DERIVATION.toCharArray(),
        SALT,
        ENCRYPTION_ITERATIONS,
        KEY_LENGTH);

    SecretKeyFactory secretKeyFactory =
        SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

    byte[] keyBytes =
        secretKeyFactory.generateSecret(secretKeySpec).getEncoded();
    SecretKey secretKey = new SecretKeySpec(keyBytes, "AES");

    return secretKey;
}
```

Listing 5.14: Derivation of an AES key using PBKDF2

5.2.3. MASWE-0011: Missing Cryptographic Key Rotation

OWASP MASWE Name	Cryptographic Key Rotation Not Implemented
Key Topics	Long-lived key, `BKS` (Deprecated Keystore), RSA
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.9.: Overview of our MASWE-0011 implementation

This vulnerability is not about adding an insecure feature but instead leaving a security feature (key rotation) out. To make the vulnerability more interesting, we added some additional security flaws to the implementation. The relevant code is shown in Listing 5.15. We use Rivest-Shamir-Adleman (RSA) for encrypting and decrypting user data. Because there is no key rotation, the RSA key will never become invalid and will not be replaced by a new and secure one. This means that if it ever falls into the hands of attackers, they could decrypt user data at will for an indefinite amount of time.

The additional security flaws consist of using the deprecated Bouncy Castle Keystore (BKS) to store our RSA key. The issues of using BKS for storing and managing our key are discussed at great length in the vulnerability implementation of Subsection 5.2.6, which is about deprecated keystores. In short; our implementation of BKS allows attackers to retrieve the private RSA key, which turns the theoretical security concern that the missing key rotation poses into a practical and detrimental security problem.

```
private static final String RSA_KEY_ALIAS = "maswe_0011_rsa_key";
private static final String KEYSTORE_FILE = "maswe_0011_keystore.bks";
private static final String KEYSTORE_PASSWORD = "Xk9$wR2!dF7pLq4Z";
private static final String KEY_PASSWORD = "S7v!Tz8#uK2qRj5M";
private static final String CIPHER_TRANSFORMATION = "RSA/ECB/PKCS1Padding";

public String encryptData(String plaintext) throws Exception {
    Cipher cipher = Cipher.getInstance(CIPHER_TRANSFORMATION, "BC");
    cipher.init(Cipher.ENCRYPT_MODE, getPublicKey());

    byte[] encrypted = cipher.doFinal(plaintext
                .getBytes(StandardCharsets.UTF_8));
    return Base64.encodeToString(encrypted, Base64.NO_WRAP);
}

private static PublicKey getPublicKey() throws Exception {
    if (keyStore == null) throw new IllegalStateException("Keystore not
        loaded");
    return keyStore.getCertificate(RSA_KEY_ALIAS).getPublicKey();
}
```

Listing 5.15: Using BKS for RSA encryption with no key rotation

5.2.4. MASWE-0012: Overloaded and Weak Cryptographic Key

OWASP MASWE Name	Insecure or Wrong Usage of Cryptographic Key
Key Topics	Overloaded Cryptographic Key, Broken Cryptographic Algorithm
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.10.: Overview of our MASWE-0012 implementation

We implemented this vulnerability by using the same cryptographic key for multiple different purposes. These include the encryption and decryption of user credentials, as well as signing and verifying messages. RSA was chosen as the cryptographic algorithm because it relies on a long-lived key that typically remains valid and in use for an extended duration. This makes it even worse if our overloaded key is compromised.

As shown in Listing 5.16, we ensured that only RSA is used for message digests involved in the signing and verification process of signatures. Furthermore, RSA with Public-Key Cryptography Standards #1 (PKCS#1) v1.5 padding is used for encrypting and decrypting data. This makes the encryption vulnerable to oracle attacks, such as the Bleichenbacher attack [54]. If this key were to fall into an attacker's hands, the attacker would be able to decrypt all existing user credentials and otherwise encrypted data. The attacker could furthermore forge message signatures and feed malicious data to the app.

```java
private static void createAndStoreKey() throws Exception {
    // ... omitted ...
    KeyGenParameterSpec spec = new KeyGenParameterSpec.Builder(RSA_KEY_ALIAS,
            KeyProperties.PURPOSE_SIGN |
                    KeyProperties.PURPOSE_VERIFY |
                    KeyProperties.PURPOSE_ENCRYPT |
                    KeyProperties.PURPOSE_DECRYPT)
            .setDigests(KeyProperties.DIGEST_SHA1)
            .setSignaturePaddings(KeyProperties.SIGNATURE_PADDING_RSA_PKCS1)
            .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_RSA_PKCS1)
            .setKeySize(2048)
            .build();
    // ... omitted ...
}
```

Listing 5.16: Creating an RSA key with multiple functionalities

5.2.5. MASWE-0014: Cryptographic Key Unprotected at Rest

OWASP MASWE Name	Cryptographic Keys Not Properly Protected at Rest
Key Topics	Lack of Encryption, `SharedPreferences`
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.11.: Overview of our MASWE-0014 implementation

For this vulnerability, we chose to use a modern and safe cryptographic algorithm, but we stored its key and initialization vector (IV) completely unencrypted in the device's `SharedPreferences`. This means that any attacker who has access to the device's files can retrieve the cryptographic key, making it completely unprotected at rest. Once the key has been retrieved, the encryptions made using it become obsolete. The relevant code for this is shown in Listing 5.17.

```java
private static void storeKeyAndIV(Context context, String encodedKey, String
    encodedIV){
    sharedPreferences = context
            .getApplicationContext()
            .getSharedPreferences(KEY_ALIAS, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();

    editor.putString(ENCRYPTION_KEY, encodedKey);
    editor.putString(IV, encodedIV);
    editor.apply();
}
```

Listing 5.17: Storing cryptographic key and initialization vector unencrypted in `SharedPreferences`

5.2.6. MASWE-0015: Use of Deprecated Keystore

OWASP MASWE Name	Deprecated Android KeyStore Implementations
Key Topics	Bouncy Castle Keystore (`BKS`), Bouncy Castle Provider (`BC`), Filebased Keystore
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.12.: Overview of our MASWE-0015 implementation

Our implementation of this vulnerability makes use of the `BKS` and the Bouncy Castle Provider (`BC`) for storing and managing the RSA key used for encrypting user credentials. The relevant code for this is shown in Listing 5.18.

```java
static {
    if (Security.getProvider("BC") == null) {
        Security.addProvider(new BouncyCastleProvider());
    }
}

public static void loadBksKeystore(Context context) throws Exception {
    if(keyStore != null){
        return;
    }

    keyStore = KeyStore.getInstance("BKS", "BC");

    try(InputStream inputStream = context.getAssets().open(KEYSTORE_FILE)) {
        keyStore.load(inputStream, KEYSTORE_PASSWORD.toCharArray());
    }
}
```

Listing 5.18: Loading `BC` and using `BKS` for managing cryptographic keys

Android considers `BKS` to be a deprecated keystore. A significant issue with `BKS` is that it is a file-based keystore, meaning that the keystore stores the keys in the `maswe_0015_keystore.bks` file, which is bundled with the APK and is easily extractable. Because of this, any attacker that downloads the app has a copy of the keystore.

The second part of this vulnerability consists of having both the keystore and the private key password hard-coded into the `EncryptionHandler.java` file, as shown in Listing 5.19. This means that an attacker who decompiles the APK has access to the file containing the private RSA encryption key used for encrypting user credentials, the password `KEYSTORE_PASSWORD` needed for opening the keystore file, as well as the `KEY_PASSWORD` needed for extracting the private RSA key. This makes it extremely easy for an attacker to gain access to decrypted user credentials.

```java
public class EncryptionHandler {
    private static final String RSA_KEY_ALIAS = "maswe_0015_rsa_key";
    private static final String SIGNING_ALGORITHM = "SHA1withRSA";
    private static final String CIPHER_TRANSFORMATION = "RSA/ECB/PKCS1Padding";
    private static final String KEYSTORE_FILE = "maswe_0015_keystore.bks";
    private static final String KEYSTORE_PASSWORD = "Xk9$wR2!dF7pLq4Z";
    private static final String KEY_PASSWORD = "S7v!Tz8#uK2qRj5M";

    // ... omitted ...
}
```

Listing 5.19: Hard-coded passwords for the keystore and private key stored in keystore

5.2.7. MASWE-0016: Unsanitized Imported Cryptographic Key

OWASP MASWE Name	Unsafe Handling of Imported Cryptographic Keys
Key Topics	Key Import From Untrusted Storage, `MediaStore`, Missing Key Sanitization, RSA
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.13.: Overview of our MASWE-0016 implementation

This implementation makes use of `MediaStore Environment.DIRECTORY_DOCUMENTS` to store and retrieve a public RSA key. The issue with exporting and importing our cryptographic key to and from `MediaStore` is that we did not add any access restrictions on the `MediaStore` entry containing the key. Because of this, other apps with the appropriate storage permissions have access to the keys placed here, enabling them to read, modify, or replace our public RSA key with a key of their own. There is, furthermore, a complete lack of identity and integrity checks conducted upon importing the key for use. If an attacking party were to replace the public RSA key with a malicious key of their own, it would be used for the future encryption of user credentials without the app noticing. Any user credentials that are encrypted using the malicious public key can then be decrypted with the attacker's private key. The app itself will start to malfunction, as it is unable to properly decrypt and log in users with its now mismatched private key.

The complete lack of sanitization and checks for the key's integrity, identity, and signature is what makes this attack possible. The import function that completely neglects to safely handle the imported key is shown in Listing 5.20.

```java
private static PublicKey importKey(Uri keyUri) throws Exception {
    InputStream inputStream =
        encryptionContext.getContentResolver().openInputStream(keyUri);
    byte[] keyBytes = readAllBytes(inputStream);

    X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
    return KeyFactory.getInstance("RSA").generatePublic(spec);
}
```

Listing 5.20: Importing a public RSA key from untrusted storage without sanitization

5.2.8. MASWE-0017: Unprotected Exported Cryptographic Key

OWASP MASWE Name	Cryptographic Keys Not Properly Protected on Export
Key Topics	No Key-Wrapping, Key Import From Untrusted Storage, `MediaStore`, RSA
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.14.: Overview of our MASWE-0017 implementation

This vulnerability is the counterpart to the vulnerability discussed in the previous Subsection 5.2.7. Instead of improperly importing a public RSA key from the `MediaStore` `Environment.DIRECTORY_DOCUMENTS`, which is an untrusted storage location that other apps can have access to, we are now improperly exporting a public RSA key to it.

The public RSA key is exported in its raw format, as shown in Listing 5.21. By implementing no key wrapping or integrity protection when exporting the key and not verifying any signatures or authenticity when importing it, we allow attackers to replace the public key with a malicious one of their own. This allows them to decrypt any user data that they manage to gather, which poses a huge security liability.

```java
private PublicKey generateAndStorePublicKey() throws Exception {
    PublicKey publicKey = getKeystorePublicKey();
    byte[] keyBytes = publicKey.getEncoded();

    ContentValues values = new ContentValues();
    values.put(MediaStore.MediaColumns.DISPLAY_NAME, PUBLIC_KEY_FILENAME);
    values.put(MediaStore.MediaColumns.MIME_TYPE, "application/octet-stream");
    values.put(MediaStore.MediaColumns.RELATIVE_PATH,
        Environment.DIRECTORY_DOCUMENTS);

    Uri uri = encryptionContext.getContentResolver().insert(
            MediaStore.Files.getContentUri("external"), values);

    OutputStream outputStream =
        encryptionContext.getContentResolver().openOutputStream(uri);
    outputStream.write(keyBytes);
    outputStream.close();

    return publicKey;
}
```

Listing 5.21: Exporting a public RSA key to untrusted storage without protecting it

5.2.9. MASWE-0018: Cryptographic Key Without Access Restrictions

OWASP MASWE Name	Cryptographic Keys Access Not Restricted
Key Topics	RSA, Key Accessible by Background Process, Key Accessible by Locked Device, Key Accessible Without Authentication
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.15.: Overview of our MASWE-0018 implementation

This vulnerability was challenging to implement due to Android security features that cannot be overridden. An intuitive way to go about this is to remove any access-restricting settings in the key generation parameters, as shown in Listing 5.22.

The issue is that this does not add a real security vulnerability to the app. Allowing access to the key with the device locked does not create any issues; Android 7.0 introduced the concept of File-Based Encryption (FBE), which partitions the device's storage into data that is available before and after booting. App data is, by default, stored in the partition that remains inaccessible until the device is fully unlocked. Because of this, an attacker cannot access any of the app's data, even if the key itself does not require the device to be unlocked to function. The authentication flag set to false runs into similar issues. It could pose a threat if the device were stolen while being unlocked. But in this case, the flag becomes redundant, as the thief would have access to the app, its data, and thus also the key, regardless of the flag. Disabling StrongBox does not create a real vulnerability, since StrongBox only adds an additional layer of physical security that is not necessary for the data storage to be considered secure.

```java
private static void createAndStoreKeyPair() throws Exception{
    // ... omitted ...

    KeyGenParameterSpec spec = new KeyGenParameterSpec
            // ... omitted ...
            .setUserAuthenticationRequired(false)
            .setUnlockedDeviceRequired(false)
            .setIsStrongBoxBacked(false)
            .setUserAuthenticationValidWhileOnBody(true)
            .build();

    keyPairGenerator.initialize(spec);
    keyPairGenerator.generateKeyPair();
}
```

Listing 5.22: Removing any access-restricting settings from key generation parameters

One way to make this vulnerability a real threat is to move the app's data to the storage partition available before booting the device. This way, the key is usable, and the user data is accessible even before the device is unlocked. A code example of this is shown in Listing 5.23. This, together with the unlocked device and user authenticated flags both set to false, could allow an attacker to use the app's cryptographic key, as well as its encryption and decryption methods on the app's data, all without the app being unlocked.

```java
Context deContext = context.createDeviceProtectedStorageContext();
SharedPreferences dePrefs = deContext.getSharedPreferences("secrets",
    MODE_PRIVATE);
dePrefs.edit().putString("encrypted_password", encryptedData).apply();
```

Listing 5.23: Storing encrypted user data in Device Encrypted (DE) storage

5.2.10. MASWE-0019: Low-Level Cryptographic Operations

OWASP MASWE Name	Risky Cryptography Implementations
Key Topics	Non-Cryptographic Functions, XOR Encryption, Use of Low-Level Mathematical Operations
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.16.: Overview of our MASWE-0019 implementation

This vulnerability has been implemented using low-level mathematical operations for the encryption and decryption of user credentials instead of proper cryptographic operations.

We used a circular bit-shifting approach for encrypting and decrypting data, as shown in Listing 5.24. For encryption, the string to be encrypted is converted into bytes. After shifting all bytes to the left by 2 bytes, any bytes that are shifted out are added back onto the right end. Afterward, the bytes are converted back to string format. The same operation, in reverse, is used for decryption.

An attacker will quickly spot the extremely weak vulnerability used here upon decompiling the app's APK. They can decrypt any user credentials by applying the decryption operation found in the code to the user credentials stored in `SharedPreferences`.

```java
public String encryptData(String plaintext) throws Exception {
    byte[] textBytes = plaintext.getBytes();

    for(int i = 0; i < textBytes.length; i++){
        textBytes[i] = (byte)((textBytes[i] << 2)
                               | ((textBytes[i] & 0xFF) >>> (6)));
    }

    return Base64.encodeToString(textBytes, Base64.DEFAULT);
}
```

Listing 5.24: Circular byte-shifting for low-level encryption of user data

5.2.11. MASWE-0020: Insecure Encryption Using Base64

OWASP MASWE Name	Improper Encryption
Key Topics	Non-Cryptographic Operations, `Base64` Encoding
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.17.: Overview of our MASWE-0020 implementation

We decided to use plain `Base64` string encoding for our implementation of improper encryption, as shown in Listing 5.25. The idea is to convert the plaintext string to bytes, which are then converted back to string format using `Base64` encoding. For decryption, this process is simply reversed. An attacker will see the weak encryption used here upon decompiling the APK, and be able to decrypt any user credentials that are stored in the `SharedPreferences` file `maswe_0020_user_credentials.xml` with ease.

```java
public String encryptData(String plaintext) throws Exception {
    return Base64.encodeToString(plaintext.getBytes(), Base64.DEFAULT);
}

public String decryptData(String encrypted) throws Exception {
    return new String(Base64.decode(encrypted, Base64.DEFAULT));
}
```

Listing 5.25: `Base64` for encrypting and decrypting user credentials

5.2.12. MASWE-0021: Insecure Hashing Using SHA-1

OWASP MASWE Name	Improper Hashing
Key Topics	SHA-1, Broken Cryptographic Algorithm
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.18.: Overview of our MASWE-0021 implementation

To implement this vulnerability, we used the Secure Hash Algorithm 1 (SHA-1) to hash user credentials for user registration and login. The concrete implementation is shown in Listing 5.26. The idea is to hash the user's password upon registration using SHA-1, then hash the attempted password during login again using SHA-1 and compare it to the existing password hashes. If there is a match of both the email and the hashed password, the user login is successful.

The vulnerability stems from the fact that SHA-1 is considered cryptographically broken, since collision attacks are practical. Google demonstrated this with the SHAttered project [55] in 2017. SHA-1 is optimized for speed, which makes brute-force attacks extremely fast.

```java
 public String hashData(String plaintext) throws Exception {
    MessageDigest digestAlgorithm = MessageDigest.getInstance("SHA-1");

    byte[] messageDigest = digestAlgorithm
                         .digest(plaintext.getBytes(StandardCharsets.UTF_8));

    BigInteger hashInt = new BigInteger(1, messageDigest);

    StringBuilder hashText = new StringBuilder(hashInt.toString(16));

    while (hashText.length() < 40) {
       hashText.insert(0, "0");
    }

    return hashText.toString();
}
```

Listing 5.26: SHA-1 hashing for user credentials

5.2.13. MASWE-0022: Hard-Coded Initialization Vectors

OWASP MASWE Name	Predictable Initialization Vectors (IVs)
Key Topics	Hard-coded IVs, AES in CBC Mode
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.19.: Overview of our MASWE-0022 implementation

IVs are meant to be random starting values used in the encryption process to ensure that two identical plaintexts still result in different ciphertexts. Since the point of this vulnerability is having predictable IVs, we decided to implement it by using a hard-coded, static IV, which breaks the layer of security added by it. The relevant parts of code are shown in Listing 5.27.

To make this vulnerability exploitable, we used the IV for encryption with AES in Cipher Block Chaining (CBC) [56] mode. Because we reuse the same hard-coded IV for all AES encryptions, the encryption becomes deterministic. If an attacker gains access to the `SharedPreferences` file containing the encrypted user passwords, they can use pattern analysis to determine how often each password is used. Furthermore, the deterministic nature of the encryption means that an attacker can identify which users have the same passwords. This is because deterministic encryption causes identical plaintexts to result in the same ciphertext. This leakage of information can improve the efficiency of brute-force attempts to steal user credentials.

```java
private static final String CIPHER_TRANSFORMATION = "AES/CBC/PKCS5PADDING";
private static byte[] iv = {47, -98, 3, 120, 14, -55, 89, 6, -12, 33, 9, -44,
    63, -1, 77, 22};

public String encryptData(String plaintext) throws Exception {
    IvParameterSpec ivSpec = new IvParameterSpec(iv);

    Cipher cipher = Cipher.getInstance(CIPHER_TRANSFORMATION);
    cipher.init(Cipher.ENCRYPT_MODE, getKey(), ivSpec);
    byte[] encryptedBytes =
        cipher.doFinal(plaintext.getBytes(StandardCharsets.UTF_8));

    return Base64.encodeToString(encryptedBytes, Base64.DEFAULT);
}
```

Listing 5.27: AES in CBC mode with a static IV

5.2.14. MASWE-0023: Padding Oracle Attacks

OWASP MASWE Name	Risky Padding
Key Topics	Padding Oracle Attacks, Manual Unpadding, PKCS#7, AES in CBC Mode
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.20.: Overview of our MASWE-0023 implementation

In Subsection 5.2.4, we implemented risky padding by using PKCS#1 v1.5 padding, which is vulnerable to the Bleichenbacher attack. We wanted to implement this vulnerability differently by using PKCS#7 padding. Since using AES in CBC mode with PKCS#7 padding is widely used and not broken in any way, we enable padding oracle attacks by manually unpadding the encrypted user data and providing detailed and varying error messages.

The relevant code for our implementation is shown in Listing 5.28. Instead of using a cipher mode for decryption that automatically removes padding, we implement our own custom unpadding process. This involves different checks, such as whether the padding length is valid or whether the padding values themselves are equal to the length of the padding. If any one of these checks fails, we throw a detailed error message. The encrypted user credentials are stored in a designated `SharedPreferences` file. If an attacker gains access to the file, they can alter the encrypted data, request the app to decrypt it, and monitor whether the padding is valid based on the response given. By repeating this process, starting from the second-to-last code block and working backward to the first, they can recalculate what the original message was.

```java
public String decryptData(String encryptedData) throws Exception {
    // ... omitted ...

    Cipher cipher = Cipher.getInstance(CIPHER_TRANSFORMATION_DECRYPTION);
    cipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec);

    if(!checkEncryptedBytesLength(encryptedBytes)){
        Log.e(TAG, "Invalid ciphertext format or length");
        return null;
    }

    if(!checkPaddingLength(paddingLength)){
        Log.e(TAG, "Invalid PKCS#7 padding length");
        return null;
    }

    if(!checkPaddingValues(decryptedBytesWithPadding, paddingLength)){
        Log.e(TAG, "Invalid PKCS#7 padding value");
        return null;
    }

    // ... omitted ...
}
```

Listing 5.28: Manual unpadding of PKCS#7 with detailed error messages

5.2.15. MASWE-0024: Insecure MAC Derivation Using CRC-32

OWASP MASWE Name	Improper Use of Message Authentication Code (MAC)
Key Topics	Non-Cryptographic Checksum, CRC-32, Missing Nonce Validation
Key Components	`IntegrityVerifier.java`, `ProfileActivity.java`, `BankAccountManagerService.java`
Programming Languages	Java

Table 5.21.: Overview of our MASWE-0024 implementation

We implemented this vulnerability by using CRC-32 [57] to authenticate and validate bank commands sent to `BankAccountManagerService.java`, which functions as a bank account manager. Instead of a cryptographically secure Message Authentication Code (MAC), bank commands receive a CRC-32 checksum. The relevant code for this vulnerability implementation is shown in Listing 5.29.

The vulnerability stems from the fact that the CRC-32 algorithm is not intended for cryptographic use and has no security properties. CRC-32 is a public algorithm that anyone can use to generate valid checksums, which would then be valid MACs for our `BankAccountManagerService.java`. An attacker who intercepts and modifies a bank command would have no issue creating a new, valid MAC for this malicious bank command. Instead of intercepting and modifying a valid bank command, an attacker can just as easily forge malicious bank commands and equip them with valid MACs via CRC-32 checksum calculation. This poses a serious security threat.

```java
private BankCommand createBankCommand(String command, int amountEuros) throws
    Exception {
    BankCommand bankCommand = new BankCommand();
    // ... omitted ...

    String payload = buildSignaturePayload(bankCommand);
    bankCommand.hmac = String.valueOf(IntegrityVerifier.crc32(payload));

    return bankCommand;
}
```

Listing 5.29: Using CRC-32 checksum as MAC

5.2.16. MASWE-0025: Insecure Signature Generation Using SHA-1

OWASP MASWE Name	Improper Generation of Cryptographic Signatures
Key Topics	Weak Signature Algorithm, `SHA1withRSA`
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.22.: Overview of our MASWE-0025 implementation

We implemented this vulnerability by using the algorithm `SHA1withRSA` to create cryptographic signatures, as shown in Listing 5.30. The issue with this lies in the fact that SHA-1 is considered cryptographically broken and insecure for signatures, as explained in the previous Subsection 5.2.12.

Using this hashing algorithm to create cryptographic signatures means that it is technically feasible for an attacker to construct a malicious message that generates the same valid hash as a real user message. This allows attackers to create fraudulent banking messages, resulting in fraudulent financial transactions that are authorized and executed by our app because of their valid signatures.

```java
public static String sign(String message) throws Exception {
    Signature signature = Signature.getInstance("SHA1withRSA");

    signature.initSign(getPrivateKey());
    signature.update(message.getBytes("UTF-8"));

    byte[] signatureBytes = signature.sign();
    return Base64.encodeToString(signatureBytes, Base64.NO_WRAP);
}
```

Listing 5.30: Creating cryptographic signatures using `SHA1withRSA`

5.2.17. MASWE-0026: Improper Signature Verification

OWASP MASWE Name	Improper Verification of Cryptographic Signature
Key Topics	Unverified Signature, Unverified Nonce, Unverified Timestamp
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.23.: Overview of our MASWE-0026 implementation

Our implementation of this vulnerability includes multiple ways in which a bank command message can be verified. This includes a nonce, a timestamp, and a Hash-based Message Authentication Code (HMAC). The code relevant for the implementation is shown in Listing 5.31.

```java
private BankCommand createBankCommand(String command, int amountEuros) throws
    Exception {
    BankCommand bankCommand = new BankCommand();
    // ... omitted ...

    bankCommand.timestamp = System.currentTimeMillis();
    bankCommand.nonce = UUID.randomUUID().toString();

    String payload = buildSignaturePayload(bankCommand);
    bankCommand.signature = EncryptionHandler.sign(payload);

    return bankCommand;
}
```

Listing 5.31: Creation of a bank command including a timestamp, a nonce and a cryptographic signature

Our implementation then goes on to ignore all three of these unique message identifiers, and verifies any bank command as valid by returning true. The timestamp, nonce and cryptographic signature are not considered at all during this verification process, as shown in Listing 5.32.

We considered implementing a more complex and less obviously flawed verification of digital signatures. However, since any flawed verification methods of higher complexity would, once understood, boil down to giving a similarly flawed return, we decided to leave it at this and focus on other vulnerabilities.

```java
public static boolean verify(BankCommand command) {
     return true;
}
```

Listing 5.32: Verifying any passed banking command as true, regardless of its timestamp, nonce and signature

5.2.18. MASWE-0027: Insecure Random Number Generation

OWASP MASWE Name	Improper Random Number Generation
Key Topics	Risky Random APIs, Non-Random Sources
Key Components	`EncryptionHandler.java`
Programming Languages	Java

Table 5.24.: Overview of our MASWE-0027 implementation

We made use of the `java.util.Random` API [58] to implement the vulnerability described by MASWE-0027. This API is not intended to be cryptographically secure. It uses a linearly congruential generator (LCG) to produce mathematically predictable random values. For this, it needs a starting seed, which we provide in the form of the system's current time in milliseconds. The code relevant for this is shown in Listing 5.33. This means that any attacker who knows the timespan during which the app was first installed can roughly guess what seed was used to create the random values. Of this estimated timespan, they only have to try 1,000 possible seeds per second. This makes finding the correct seed and the generated random values highly feasible.

Because we use these random values to generate the IV, and because we reuse this IV for AES in CBC mode, this poses a serious security threat. Reusing the same IV for AES in CBC mode makes the encryption deterministic. An attacker who reconstructs the generated random values used for the IV can analyze which ciphertexts stem from the same plaintext. This enables password frequency analysis, which can result in compromised user credentials.

```java
import java.util.Random;

public class EncryptionHandler {
    private static final String ENCRYPTION_ALGORITHM = "AES";
    private static final String CIPHER_TRANSFORMATION = "AES/CBC/PKCS5PADDING";
    // ... omitted ...

    private static String createIV(){
        byte[] iv = new byte[16];
        Random javaRandom = new Random(System.currentTimeMillis());
        javaRandom.nextBytes(iv);

        String encodedIV = Base64.encodeToString(iv, Base64.DEFAULT);
        return encodedIV;
    }
    // ... omitted ...
}
```

Listing 5.33: Using `java.util.Random` API and current system time for generating random values

5.3. MASVS-PLATFORM

5.3.1. MASWE-0053: Password Leaked Through Input Fields

OWASP MASWE Name	Sensitive Data Leaked via the User Interface
Key Topics	Missing Password Obfuscation, Password Autocomplete, Password Autocorrect, Password Saved to Clipboard
Key Components	`LoginActivity.java`, `RegisterActivity.java`
Programming Languages	Java

Table 5.25.: Overview of our MASWE-0053 implementation

To introduce the risk of the users themselves leaking sensitive data, we removed any text obfuscation on input fields for user passwords both in `LoginActivity.java` and `RegisterActivity.java`. The code for this is shown in Listing 5.34. Additionally, we disabled all screenshot and screen recording protections.

By enabling copy and paste functionalities for the password field, any text that is copied or cut from the input field will be stored in the `Android Clipboard`. Data stored

```java
private void removeObfuscation(){
    EditText email_field = findViewById(R.id.et_email);
    EditText password_field = findViewById(R.id.et_password);

    password_field.setInputType(InputType.TYPE_CLASS_TEXT |
            InputType.TYPE_TEXT_FLAG_AUTO_COMPLETE |
            InputType.TYPE_TEXT_FLAG_AUTO_CORRECT);

    email_field.setInputType(InputType.TYPE_TEXT_FLAG_AUTO_COMPLETE |
            InputType.TYPE_TEXT_FLAG_AUTO_CORRECT);

    password_field.setCustomSelectionActionModeCallback(null);
}
```

Listing 5.34: Removing text obfuscation and adding auto complete and auto correct flags for user input fields

in the `Android Clipboard` is not confined to the app's sandbox and is accessible by all apps running in the foreground. This holds true even on newer Android versions [59]. This means that an attacker can run a service from a malicious app installed on the device that accesses data from the `Android Clipboard`, potentially reading and stealing sensitive user data.

5.3.2. MASWE-0055: Password Exposed to Screenshots

OWASP MASWE Name	Sensitive Data Leaked via Screenshots or Screen Recordings
Key Topics	Missing Password Obfuscation, Missing Screenshot Protection
Key Components	`LoginActivity.java`, `RegisterActivity.java`, `activity_login_maswe0055.xml`, `activity_register_maswe0055.xml`
Programming Languages	Java, XML

Table 5.26.: Overview of our MASWE-0055 implementation

To implement this vulnerability, we ensured that the password is not obfuscated. Additionally, we allowed screenshots and screen recordings of the activities where the password is visible. To differentiate this vulnerability from the implementation in the previous Subsection 5.3.1, the password obfuscation is removed in the layout itself, instead of changing the password's type to plain text at runtime. The code for this is shown in Listing 5.35.

```
<EditText
    android:id="@+id/et_password"
    <!-- ... omitted ... -->

    android:hint="@string/password_hint"
    android:inputType="textVisiblePassword" />
```

Listing 5.35: Setting the password `EditText` to visible password as to remove text obfuscation

We furthermore disable and clear any lingering any flags that would prevent screenshots and screen recordings from happening, as shown in Listing 5.36. This implementation introduces the security risk of the user leaking sensitive data via accidental screenshots and screen recordings. There are also tools that allow an attacker to record the device's screen via a USB connection, such as `scrcpy` [60].

```
private void clearScreenshotFlags(){
    /**
     getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
     WindowManager.LayoutParams.FLAG_SECURE);
     **/
    getWindow().clearFlags(WindowManager.LayoutParams.FLAG_SECURE);
}
```

Listing 5.36: Disabling and clearing security flags that prevent screenshots and screen recordings from login and register activities

5.3.3. MASWE-0064: Insecure Content Provider

OWASP MASWE Name	Insecure Content Providers
Key Topics	Exported Content Provider, Insecure File Reading Permissions, Unsanitized Path Construction, Missing Access Control
Key Components	`CustomContentProvider.java`, `RegisterActivity.java`, `AndroidManifest.xml`
Programming Languages	Java, XML

Table 5.27.: Overview of our MASWE-0064 implementation

The core of this vulnerability's implementation comes from the code snippet of our custom content provider shown in Listing 5.37. The vulnerability comes from the fact that `uri.getLastPathSegment()` extracts the user-given file `URI` without any validation in place. The content provider tries to find the file, construct its full path and return a file descriptor granting reading permissions to the file. For all of this, the content provider directly uses the unsanitized user input.

```
@Override
public ParcelFileDescriptor openFile(Uri uri, String mode) throws
    FileNotFoundException {
    String fileName = uri.getLastPathSegment();

    if (fileName == null) {
        throw new FileNotFoundException("No file name specified in URI");
    }

    File file = new File(getContext().getFilesDir(), fileName);
    return ParcelFileDescriptor.open(file, ParcelFileDescriptor.MODE_READ_ONLY);
}
```

Listing 5.37: Custom `ContentProvider` granting reading permission to user-requested file without path traversal checks or sanitizing the input

This allows for path traversal attacks by using path traversal sequences such as `../`. Without it, the content provider would only be able to provide access to files stored within the `/files/*` directory. This way, an attacker can access and read any of the app's files, given that they have access to the custom content provider. To ensure this is the case and further extend the impact of this vulnerability, the content provider is set to be exported in the `AndroidManifest.xml` of this app, as shown in Listing 5.38. By leaving the `permission` parameter empty, we enforce that no permissions are required by other apps to access this app's content provider.

```
<provider
    android:name="com.dkronig.masvs_platform.maswe_0064.CustomContentProvider"
    android:authorities="com.dkronig.masvs_platform.CustomContentProvider"
    android:exported="true"
    android:permission=""
    tools:ignore="ExportedContentProvider" />
```

Listing 5.38: Exporting the custom `ContentProvider`, making it accessible to other apps without enforcing permissions

This, in combination with the unsanitized content provider, means that any malicious app installed on the same device can trivially access any of the app's files, either directly or via path traversal. This is demonstrated in Listing 5.39.

```
// Direct file access
Uri.parse("content://com.dkronig.masvs_platform.CustomContentProvider
    /maswe_0064_user_credentials.txt");
InputStream in = getContentResolver().openInputStream(uri);

// Read files via path traversal
Uri uri = Uri.parse("content://com.dkronig.masvs_platform.CustomContentProvider
    /../databases/app.db");
InputStream in = getContentResolver().openInputStream(uri);
```

Listing 5.39: Accessing and reading files containing sensitive user data from malicious apps, both directly and using path traversal

5.3.4. MASWE-0067: App Set to Debuggable

OWASP MASWE Name	Debuggable Flag Not Disabled
Key Topics	Memory Inspection, Cryptographic Key Access
Key Components	`AndroidManifest.xml`
Programming Languages	XML

Table 5.28.: Overview of our MASWE-0067 implementation

The implementation of this vulnerability is rather straightforward, as only one flag had to be set to true in the `AndroidManifest.xml` for the app to be debuggable. The code for this is shown in Listing 5.40.

```
<application
    android:theme="@style/Theme.masvs_platform"
    <!-- ... omitted ... -->

    android:debuggable="true"
    tools:ignore="HardcodedDebugMode">
```

Listing 5.40: Setting the app to debuggable in `AndroidManifest.xml`

The issue with the app set to debuggable is that it directly weakens the app's protection. It allows an attacker to inspect, set breakpoints, control, and modify the app at runtime. An attacker can furthermore access the app's private system directory `/data/data` by using the `run-as` command. Using this command, an attacker can inspect the app's `SharedPreferences`, SQLite databases, and internal storage at run-time. An attacker can, for example, learn more about the encryption in place for user credentials by creating new user profiles and analyzing the resulting ciphertexts. An example of how an attacker can access the user credentials stored in a `SharedPreferences` file is shown in Listing 5.41.

```
C:\Users\Domi>adb shell run-as com.dkronig.masvs_platform ls shared_prefs
maswe_0053_user_credentials.xml
maswe_0055_user_credentials.xml
maswe_0064_user_credentials.xml
maswe_0067_user_credentials.xml

C:\Users\Domi>adb shell run-as com.dkronig.masvs_platform cat
    shared_prefs/maswe_0053_user_credentials.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="users_json">{&quot;qwertz&quot;:{&quot;password&quot;
        :&quot;doVMBwPTG2ISsgy06P\/eJpfWXKT5dOJIO9jYl8FzWhGfmA==&quot;}}</string>
</map>
```

Listing 5.41: Listing and reading `SharedPreferences` files containing user credentials

6. Results

This chapter describes the MASTG reference apps we designed and implemented, as well as the MASWE vulnerability types they include. A comparison is made with the existing reference apps, and both the total coverage of MASWE vulnerability types across all apps and the coverage per individual app are evaluated. To conclude the chapter, we test two different SAST tools on our apps, and cover the most important findings.

6.1. Developed MASTG Reference Applications

We developed three separate apps that are named `masvs_storage`, `masvs_crypto`, and `masvs_platform`. They contain a total of 28 vulnerabilities. All six vulnerabilities described by MASVS-STORAGE and all 18 MASVS-CRYPTO vulnerabilities are covered. The remaining four implemented vulnerabilities are part of MASVS-PLATFORM. This means we created implementations for 28 out of 115 vulnerability types as documented by OWASP MASWE, covering about 25% of the entire vulnerability catalog. Excluded here are the two vulnerability types that the OWASP MASWE lists as deprecated. Each vulnerability has its own documentation, explaining where it can be found in the code and how it can be exploited. Depending on the vulnerability, the documentation also includes links to interesting and relevant articles and tools related to the vulnerability. The documentation of some vulnerabilities contains descriptions of how the vulnerabilities can be fixed. All implemented vulnerabilities are mapped and linked to their respective OWASP MASWE counterparts.

The code for our apps is entirely open source and available on our GitHub repository [51]. It contains a checklist [61] of all vulnerabilities as described by OWASP MASWE, which provides an overview of the vulnerabilities implemented. Furthermore, it indicates the vulnerabilities that are being worked on, as well as those that have yet to receive attention. Every vulnerability in this overview has an indicator of its status in OWASP MASWE. Most MASWE vulnerability types have the status `placeholder`, indicating that the vulnerability is still in development and lacks proper documentation by OWASP. The rest is marked as fully documented but still in `beta`, with the exception of two flagged as `deprecated`, indicating that they should no longer be considered. This benefits people using the apps and the GitHub repository for educational purposes by allowing them to better understand the complete picture of the MASWE vulnerability landscape. It further helps developers aiming to contribute to this project, as it points them to which vulnerabilities are yet to be worked on, and which should be ignored. To further enable future work on this project, the repository is complete with documentation on how developers can contribute, what the expected coding styles are, and what templates should be used when committing to the project. This enforces a clean and uniform coding style across the project. This further improves its usefulness as an educational tool by offering people who are learning security principles or penetration testing an environment free of confusing and inconsistent code.

For the documentation and the vulnerabilities themselves to stay relevant, and for the apps as a whole to be a valid reference resource for the OWASP MASTG, constant maintenance and updating of the Android project are necessary. Without it, our apps are exposed to the risk of becoming outdated and no longer accurately reflecting the vulnerabilities discussed by the OWASP MAS project.

6.2. Comparison with Existing Intentionally Vulnerable Android Applications

6.2.1. Total Vulnerability Coverage

It is important to compare the apps we have developed to the existing pool of MASTG reference apps. This way, we can evaluate the contributions and value generated through the addition of our apps to the MASTG reference apps dataset. An intuitive way of doing so is to compare the coverage of different MASVS categories by all apps, both with and without the addition of our apps. Figure 6.1 shows the percentage-wise coverage of different MASVS categories by all apps, including our own. Comparing this with Figure 2.2, which displays the same but excludes our own apps, we can see the contribution made in terms of MASVS categories coverage. This difference made by including our apps is further visualized in Figure 6.2.

The biggest contribution of our apps is to the MASVS-CRYPTO category. The number of MASVS-CRYPTO vulnerabilities that have a practical reference implementation in the MASTG reference apps has been increased from 21.1% to 100%. This makes sense, as 18 of our 28 implemented vulnerabilities are intended to reflect MASVS-CRYPTO vulnerabilities. In a similar fashion, the coverage of MASVS-STORAGE vulnerabilities

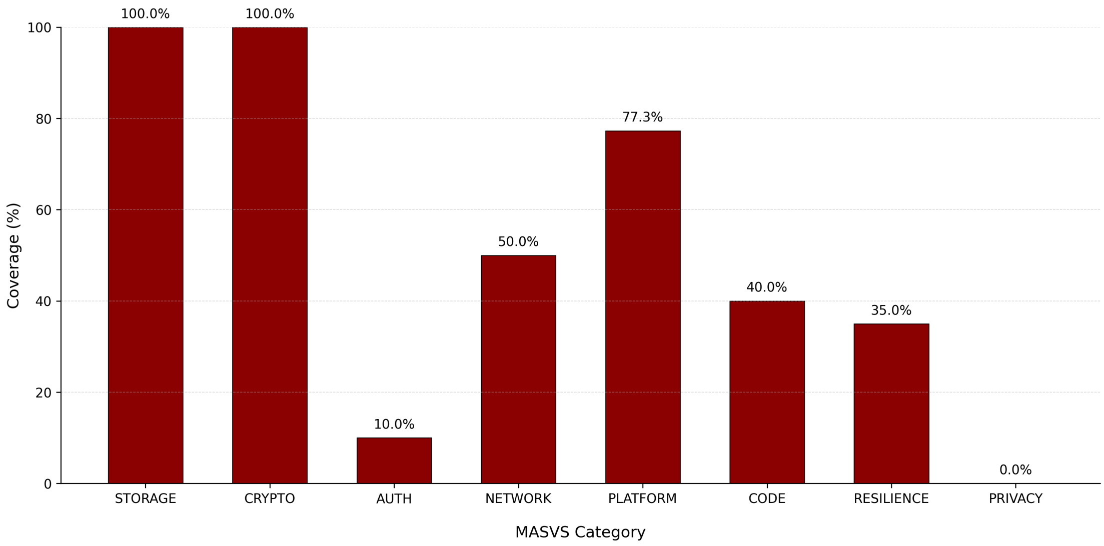


Figure 6.1.: Relative coverage of MASWE vulnerabilities including our own apps

has been increased from 66.7% to 100%. And lastly, the number of MASVS-PLATFORM vulnerabilities that are reflected in an implementation has been increased from 72.7% to 77.3%. This strongly concentrated effect of contribution reflects the vulnerability selection criteria we had set for ourselves, focusing on a few categories and implementing their vulnerabilities thoroughly. This has achieved the desired effect of making our `masvs_crypto` and `masvs_storage` apps valuable and useful standalone apps, both for evaluating security analysis tools and for learning purposes.

Figure 6.3 shows the same coverage of MASWE vulnerabilities by all apps, including our own, but uses absolute numbers instead of percentages. By comparing it to Figure 2.3, which shows the same results but excludes our own apps, we can again see the significantly improved and complete coverage of both MASVS-STORAGE and MASVS-CRYPTO vulnerabilities. Complete coverage here does not mean that all possible implementations of these vulnerabilities have been created, and that further such implementations would hold no value. It simply means that all vulnerability types in the categories MASVS-STORAGE and MASVS-CRYPTO now have at least one vulnerability implementation representing them in a practical context. The Figure further shows that although both categories now have a coverage of 100%, the contribution to MASVS-CRYPTO is significantly greater than the one to MASVS-STORAGE. This is because there are only 6 storage vulnerabilities required to cover them all, compared to

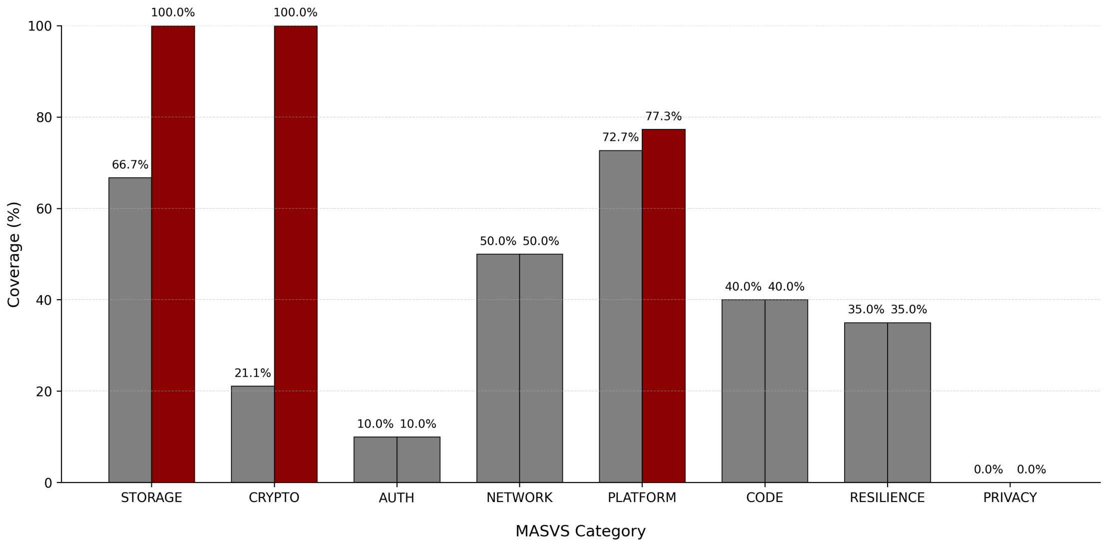


Figure 6.2.: Contribution of our apps (in red) to the relative coverage of MASWE vulnerabilities

the 18 vulnerabilities described by MASVS-CRYPTO. Here we are excluding the one MASVS-CRYPTO vulnerability that is flagged as deprecated by OWASP.

6.2.2. Unique Vulnerability Coverage per Application

One of our criteria defined for the selection of vulnerabilities was to focus on vulnerabilities which have seen little to no coverage in existing MASTG reference apps. The purpose of this is to broaden the pool of MASWE vulnerabilities with practical reference implementations. Of our 28 implemented vulnerabilities, 18 have not been implemented in any way by the existing reference apps. This means that 64.3% of our implemented vulnerabilities are of MASWE vulnerability types only found in our MASTG reference apps. This is not to imply that the other vulnerabilities implemented are not unique in their own right. Even if two implementations address the same vulnerability, they will most likely still differ greatly in logic and design. It does mean however that 64.3% of the vulnerability types we included in our apps have not seen any coverage in any of the existing MASTG reference apps. This approach was intended to even out the coverage of vulnerabilities. The reason for this was our observation that more than half of all vulnerabilities implemented are a reference to only ten OWASP MASWE vulnerability types. Without our apps, the MASTG dataset includes 163 vulnerability implementations, which can be mapped to only 42 different MASWE vulnerabilities. This means

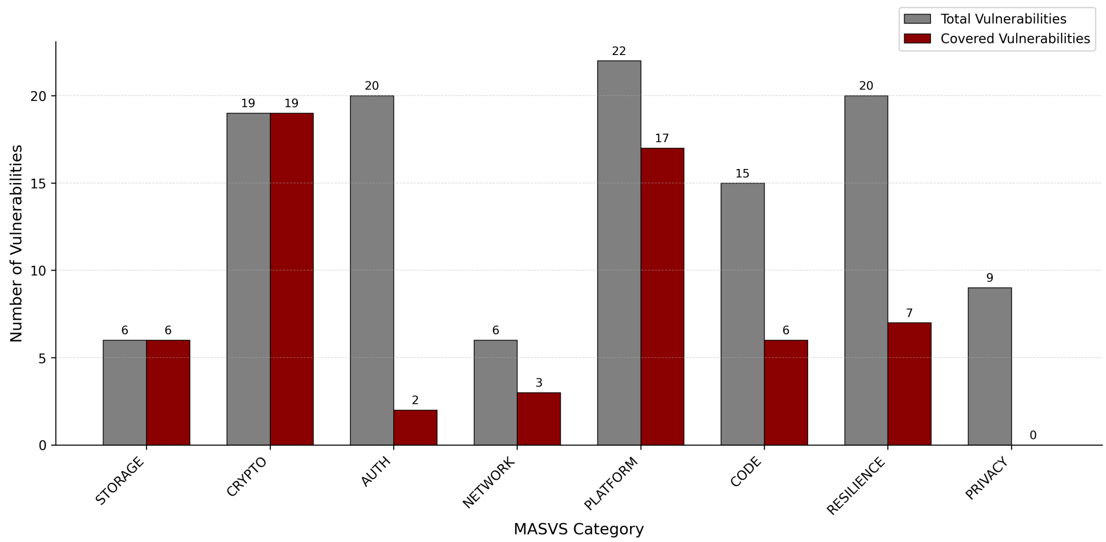


Figure 6.3.: MASWE vulnerabilities and their absolute coverage by reference apps

that only 25.8% (42/163) of all vulnerabilities found in the existing MASTG reference apps contribute to the coverage of different MASWE vulnerability types. Our own apps stand in stark contrast to this number, with a ratio of 64.3% (18/28). Figure 6.4 illustrates this by showing the total amount of MASWE vulnerabilities covered per app that do not see coverage in any of the other existing apps in the dataset.

This furthermore means that our own apps improve the total coverage of OWASP MASWE vulnerabilities from 42 to 60 out of 117, resulting in an increase of coverage from 35.9% (42/117) to 51.3% (60/117). The percentage of implementations that contribute to the coverage of different MASWE vulnerability types has increased from 25.8% (42/163) to 31.4% (60/191).

If we add our own apps to the previous graph, as shown in Figure 6.5, the result of our focus on vulnerability types with no previous coverage becomes clear. Figure 6.6 better illustrates this by adding the total number of implemented vulnerabilities to each app.

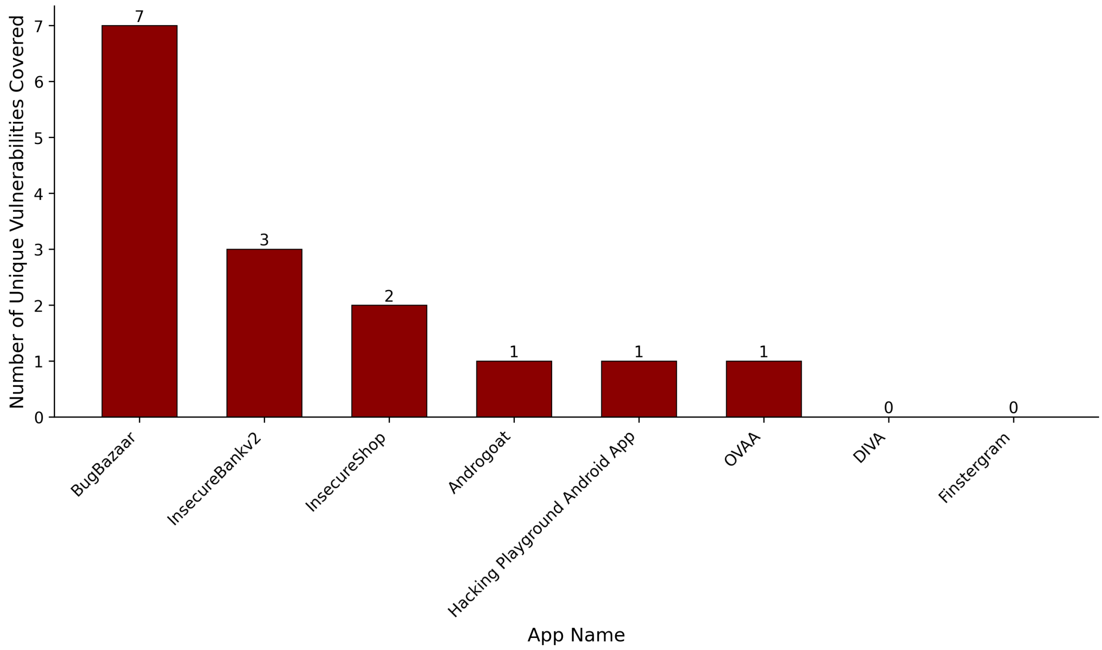


Figure 6.4.: Amount of MASWE vulnerabilities covered per app which are not implemented by other MASTG reference apps

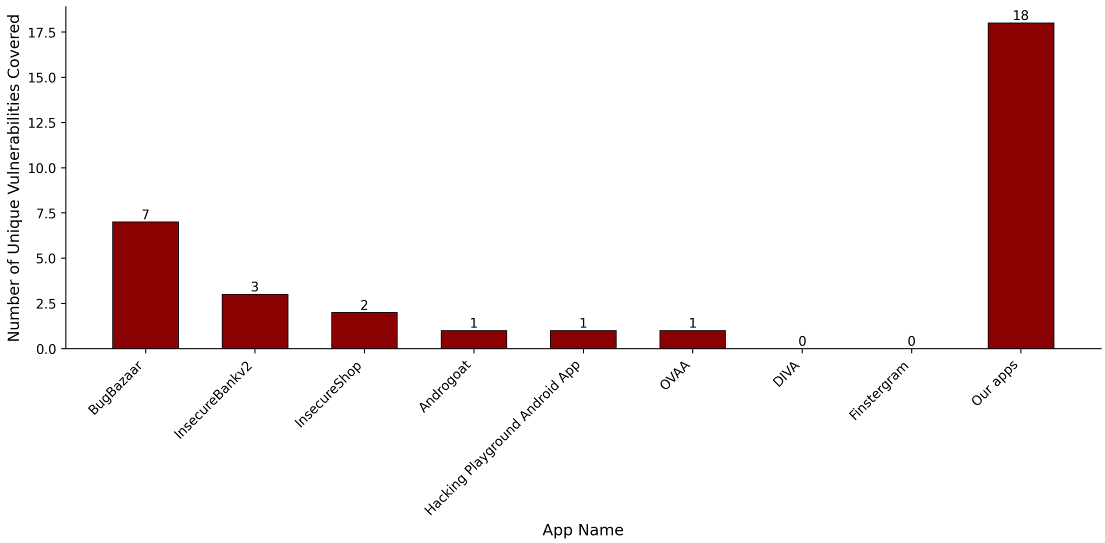


Figure 6.5.: Amount of MASWE vulnerabilities covered per app which are not implemented by other MASTG reference apps, including our own apps

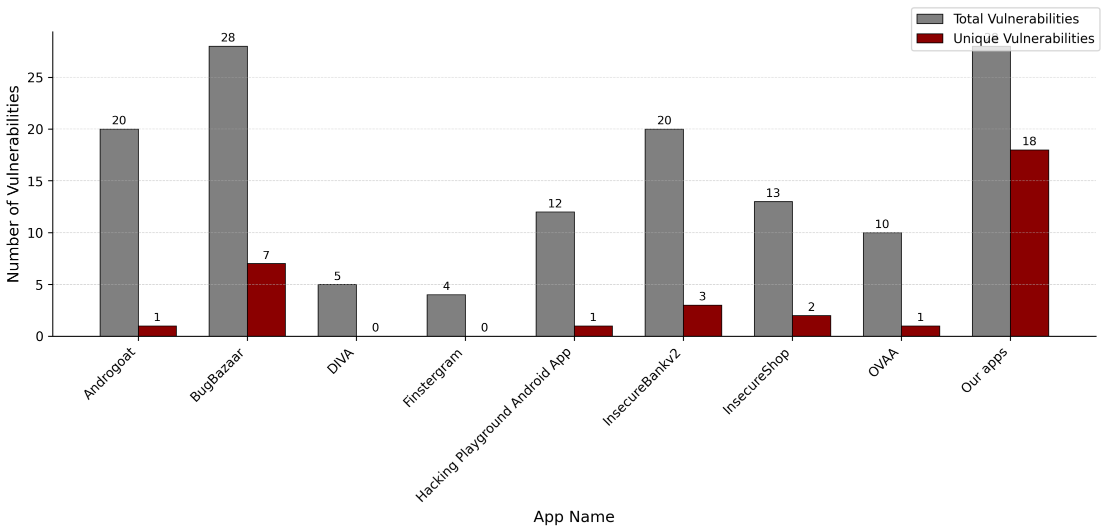


Figure 6.6.: Ratio of MASWE vulnerabilities to amount of vulnerabilities implemented per app

6.3. Usability for Benchmarking Mobile App Security Testing Tools

A key motivation behind our work is the testing and benchmarking of SAST and DAST tools. Although our focus is on the design and development of MASTG reference apps, benchmarking different security testing tools using our apps would be a natural direction to take for future work. To take a first step in this direction, we used an online deployment of MobSF [62] and a free version of Semgrep [63] to statically evaluate our apps for security vulnerabilities. The complete results for both the MobSF and the Semgrep scans are provided in the Appendix A and B.

MobSF Results

Interestingly, both `masvs_storage` and `masvs_crypto` received a low risk grade by MobSF, having security scores of 64/100 and 60/100 each. `masvs_platform` was deemed to be of medium risk and received a security score of 52/100, which marks the lowest of all three.

In terms of storage vulnerabilities, MobSF managed to find the hard-coded cryptographic key in `maswe_0006` and detected the logging of sensitive user data in `maswe_0001`.

However, it failed to understand the risk of storing sensitive user data in shared storage spaces and was unable to grasp the full extent of the backup vulnerabilities.

Of 18 implemented MASVS-CRYPTO vulnerabilities, MobSF managed to find the insecure random number generator and the usage of logs for enabling a padding oracle attack. It detected one implementation of SHA-1, and nine instances of hard-coded sensitive information. It is interesting that it found and flagged one instance of SHA-1, but failed to do so on the many other instances of SHA-1 across the app. It furthermore ignored all other broken or risky forms of cryptographic algorithms, such as DES, Base64 encoding, or AES in GCM or CBC mode with hard-coded IV's. These are all things that can be statically evaluated and detected.

The low security score of `masvs_platform` is a result of the app including a vulnerability that sets it to debuggable. MobSF did manage to find this vulnerability, as well as the insecure custom content provider. It did not find the vulnerabilities enabling the screenshots of sensitive user data, or simply did not consider it to be a real security threat. The same holds for the vulnerability enabling clipboard attacks for stealing user credentials.

A large part of the scan results by MobSF consists of potential hard-coded secrets that are false positives. They are public parameters used by cryptographic algorithms and libraries and do not contain actual secrets. An example of this is `6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296`, which marks the x-coordinate of the generator point G of the NIST P-256 elliptic curve. This is a publicly documented value from FIPS 186-4 and is in no shape or form a hard-coded secret [64].

In summary, MobSF gave our intentionally vulnerable apps a surprisingly high security score. Of the 28 vulnerabilities implemented across the three apps, it managed to clearly identify 8 of them. It failed to detect many instances of deprecated cryptographic algorithms used, as well as all instances of custom low-level cryptographic algorithms. Besides failing to find many of our cryptographic vulnerabilities, MobSF flagged a large number of cryptographic constants as possible hard-coded secrets, leading to many false positives.

Semgrep Results

For Semgrep, we used the free version on our apps, with the configurations set to `r/java.lang.security`. This scanned all three apps for static security vulnerabilities. It is important to state that Semgrep has more features and configurations that would lead to improved performance. We decided to focus on the free version for monetary reasons.

Because of our configuration, Semgrep only found MASVS-CRYPTO vulnerabilities. It found 19 instances of vulnerable code across 8 of our implementations. Semgrep was able to detect more complex vulnerabilities than MobSF. Besides flagging the usage of SHA-1 as insecure, it did the same for any implementations that used DES. It detected hard-coded IV's and the combination of this with AES in GCM mode. However, it failed to mark any of the low-level cryptographic operations used for encryption as insecure, similar to MobSF.

7. Discussion

Here we discuss possible threats to the validity of our work. These include the often limited descriptions of the vulnerabilities by OWASP MASWE and the possibility of the implemented vulnerabilities failing to mirror real-world equivalents. Furthermore, we discuss the risk of the implemented vulnerabilities having insufficient complexity and depth compared to those found in real-world applications. Following this, problems left open by this thesis are addressed, such as what it means to fully cover a MASVS category and what a valid metric for measuring the value presented by a MASTG reference app could look like.

7.1. Threats to Validity

7.1.1. Constant Flux of the Vulnerability Landscape

A challenge to our work was the ever-changing nature of the mobile vulnerability landscape and its impacts on the OWASP MAS project. Because OWASP is constantly adapting to the rise of new vulnerabilities and the obsolescence of older ones, their vulnerability documentation is a permanent construction site. In an effort to keep up with real-world changes, older vulnerabilities are overworked, merged together into new ones, or altogether scrapped and considered deprecated. New vulnerabilities are introduced into the MASWE catalog in the form of placeholder vulnerabilities. These placeholder vulnerabilities contain very limited documentation on what they should look like, how they are usually introduced, and what they should entail. At the time of writing, 88 of 117 OWASP MASWE vulnerability types fall into this placeholder category [65]. Because two vulnerability types are further classified as deprecated, this leaves only 27 vulnerabilities that have proper and complete documentation by MASWE.

Only 10 of the 28 vulnerabilities implemented in our apps have complete documentation by OWASP MASWE, with the remaining 18 still in the placeholder phase. This raises the question of whether our implementations accurately reflect the MASWE vulnerability types that they are supposed to serve as a practical reference for. We kept this concern in mind during the development phase of the apps and made sure to consult the links the MASWE gives as pointers to the nature of the vulnerabilities. These resources are provided for all placeholder vulnerabilities, which counteracts the lack of real documentation on them by MASWE.

7.1.2. Insufficient Reflection of Real-World Vulnerabilities

A point of criticism often raised when discussing intentionally vulnerable Android apps, such as those found in the MASTG reference apps catalog, is about the extent to which these apps accurately reflect the vulnerabilities found in real-world Android apps. Especially when using them for benchmarking SAST and DAST tools, which are then used for testing real-world apps, it is important for the benchmarking apps to closely mimic real-world ones. If this is not the case, they lose their function for evaluating such security tools, since a low or high score on the reference apps would not necessarily translate to a similar performance on real apps. The reference apps might include vulnerabilities which are rarely seen in real-world apps, while missing crucial vulnerabilities that a tool has to be able to detect.

Speaking against this point of criticism is the relevance and status of the OWASP MASTG as an industry standard for evaluating mobile application security. There are numerous papers [1, 25, 27, 28, 30, 34] that show the effectiveness of the MASTG when it comes to finding vulnerabilities. Anwar et al. [26] even found that using the MASTG leads to finding more vulnerabilities in real-world apps than using the automated testing tool MobSF. This solidifies the legitimacy and real-world application of the vulnerabilities implemented in the MASTG reference apps, since the MASTG concerns itself with the same vulnerabilities that are discussed in the MASWE and found in the reference apps. For this to be case however, it is crucial that the MASTG reference apps are under constant maintenance and evolve alongside the OWASP MAS project, which in turn adapts to the mobile security issues most prevalent at the time.

7.1.3. Insufficient Complexity of Vulnerabilities

Besides not reflecting the vulnerabilities found in real-world apps, it is possible that the vulnerabilities implemented in the MASTG reference apps are of the right type, but are implemented in a shallow and unrealistic way, which fails to reflect how the vulnerabilities look like in real-world apps. This would lead to mobile app security testing tools performing well on the MASTG reference apps due to the easy-to-spot nature of their implemented vulnerabilities, but struggling to do the same on real-world apps that include similar yet more complex vulnerabilities.

To counter this argument, a study would have to be conducted, collecting vulnerabilities from real-world apps and comparing them to the ones found in the MASTG reference apps catalog. Zhu et al. [35] evaluated the effectiveness of 11 different SAST tools on four different benchmarks. Two of them are synthetic, and the other two are real-world benchmarks based on vulnerabilities found in real-world apps. One of the synthetic benchmarks is based on the OWASP MASTG reference apps catalog. The authors assess that there is no significant difference in the performance of the 11 SAST tools between the synthetic and the real-world benchmarks. This illustrates that the vulnerabilities implemented in MASTG reference apps are similar in both type and complexity to the ones found in real-world apps.

7.1.4. Relevance of Even MASWE Vulnerability Coverage

While deciding which MASWE vulnerabilities to implement, we focused on vulnerabilities that have not been implemented by other MASTG reference apps. Our rationale behind this decision was that all vulnerabilities described by the OWASP MASWE are relevant. Because of this, they should all have some form of representation in the reference apps catalog, both for educational purposes and for evaluating security testing tools. An argument can be made against the validity of this approach, suggesting to focus on the more frequently implemented vulnerabilities instead of the opposite group. It can be argued that there is a reason as to why certain vulnerabilities receive most of the attention, which is that they are simply the most relevant vulnerabilities. The distribution of occurrences of different vulnerability types is likely uneven in the real world, with some happening more often than others. With this being the case, it seems intuitive that the reference apps, which to a certain extent should mirror the vulnerabilities found in the real world, would also have an uneven distribution of vulnerability types.

Although this argument makes sense, it does not address the issue of most MASWE vulnerabilities receiving no coverage at all. Even if uneven coverage of the vulnerabilities is desired, it is still important that all vulnerabilities have at least one practical reference implementation. To focus on mirroring the real-world distribution of vulnerability type occurrences before having all the MASWE vulnerabilities implemented at least once seems premature and the wrong criterion for vulnerability selection.

7.2. Open Problems

Through the developed apps, we increased the coverage of MASWE vulnerability types for both the storage and the cryptography categories to 100%. What we mean by this is that we created a reference implementation for every vulnerability type listed in these two categories. It does not mean that we implemented every possible vulnerability of these two categories. But what would it take to actually cover a MASVS category to 100%? It is important to discuss what it means for two implementations to reflect the same vulnerability, and what it would take to truly cover a MASVS category completely.

For many vulnerabilities described by MASWE, there are countless ways to implement them. This is because they describe types of mobile security shortcomings, and just like in the real world, there are often multiple ways in which one can fall short of meeting a security criterion. This means that vulnerability implementations that can be mapped to the same MASWE vulnerability can still differ substantially in form. Because of this, the educational and security testing tool value of a MASTG reference app can be increased by repeatedly developing implementations for MASWE vulnerabilities that have been previously covered. The same holds true if the vulnerability category itself has been fully covered by this app or other ones from the MASTG reference apps catalog. This raises multiple questions. Is the coverage of vulnerability types of MASVS categories a valid metric for measuring the value a reference app provides? And following this line of thought, does it even make sense to aim for the complete coverage of MASVS categories?

Answering these questions can help us find a useful metric for evaluating the contributions made by a MASTG reference app, as well as where its shortcomings lie, and how it can be further developed. For this, it is important to keep in mind the purpose of the OWASP MASTG and how it functions. The OWASP MASTG contains various knowledge areas on mobile app security testing concepts, techniques, tools, and best practices. Its main function stems from the catalog of tests that it provides. These can be used to verify the security controls listed by the MASVS by testing for the vulnerabilities described by the MASWE. These tests are concrete ways in which developers can evaluate their own apps for certain implementations of MASWE vulnerabilities. The vulnerability implementations that the MASTG tests for are defined by the OWASP MAS project. This stands in contrast to the MASTG reference apps, which are merely hosted but not developed by OWASP. These implementations give a clue to how OWASP MAS would implement the vulnerabilities described by the MASWE. Furthermore, since the reference apps are intended to serve as reference apps for the OWASP MASTG, it seems logical for the reference apps to cover the vulnerability implementations found in the MASTG tests. This way, the apps function as a resource on which the concrete MASTG tests can be conducted.

Following this line of thinking, a possible metric for evaluating MASTG reference apps is to measure the extent to which they cover or contribute to the total coverage of vulnerability implementations found in the MASTG tests. By focusing on reflecting the material of the actual tests of the MASTG, the apps are ensured to provide real value as a reference resource for the testing guide. Some of the tests already include demos made by OWASP that display the vulnerability that the test is designed to find. But the catalog of demos is incomplete, and the demos focus solely on the vulnerability. Because of this, they lack elements of real-world applications that many of the MASTG reference apps include. Furthermore, the design of the MASTG tests allows each tested vulnerability to have multiple adequate implementations. This means that creating different vulnerability implementations for the same MASTG test cases can still provide new value, even if the test has an existing demo.

This approach helps evaluate whether a new implementation of an already covered vulnerability type is of value. But it leaves open the question of what it would mean for a vulnerability category to be covered completely. It seems that complete coverage of a vulnerability category is not feasible. Because of this, it should not be the goal of and metric for a valuable MASTG reference app. The focus of vulnerability selection should first be on covering all vulnerability types at least once. Once this has been achieved, there is merit in creating different vulnerability implementations of the same MASWE vulnerabilities. This way, the OWASP MASTG reference apps can be continuously improved and expanded upon.

8. Future Work

While the apps we developed significantly increased the coverage of different MASWE vulnerability types, there are still many such vulnerability types that lack a reference implementation. 57 MASWE vulnerability types do not have any representation in the MASTG reference apps, including our own. This means that only 51.3% (60/117) of all vulnerability types are included in the apps catalog. Our work can be expanded upon by moving on to new MASVS categories, designing and implementing vulnerabilities in a way similar to our methodology. The categories MASVS-AUTH and MASVS-PRIVACY are of special interest. They have the lowest ratio of described-to-implemented vulnerabilities, with only 2 out of 20 and 0 out of 9 vulnerability types, respectively, receiving any coverage in the reference apps.

Our work is not only motivated by improving the educational value of the OWASP MASTG reference apps, but also their ability to be used for benchmarking SAST and DAST tools. We have tested both MobSF and Semgrep for their performance in statically evaluating our apps. However, benchmarking the different tools that comprise the modern mobile testing tools landscape lies outside the scope and resources of our work. A natural next step would be to carry out a scientific evaluation and benchmarking process of modern mobile app testing tools. This should include tools of both static and dynamic nature. Furthermore, their performance on our apps can be compared to their performance on other MASTG reference apps, as well as against other existing mobile security tool testing benchmarks. This way, the value that our developed apps hold in terms of testing such tools can be measured.

The Android application landscape and its vulnerabilities are constantly evolving, and the OWASP MASTG is evolving alongside it. Therefore, an important next step is to uphold the maintenance of our apps. The MASWE vulnerabilities and the MASTG tests are constantly being deprecated and replaced with new ones. Even the MASVS categories themselves can change, with the most recent addition being MASVS-PRIVACY [24]. Our apps need to receive constant updates because the MASTG reference apps can only retain their value both as a learning resource and as a dataset for benchmarking mobile security testing tools if they keep up with the ever-changing Android vulnerability landscape.

The OWASP MAS project does not concern itself exclusively with Android security. It includes a similar segment dedicated to the mobile security of iOS applications as well. A common belief is that iOS apps have better security standards due to the more regulated nature of the app platforms from which they are distributed. There are, however, many security flaws and vulnerabilities that iOS apps can include, as described by the related OWASP MAS fields. Steinböck et al. [38] showed that iOS apps perform significantly worse in certain security categories when compared to Android apps. Furthermore, the dataset of MASTG reference apps for Android [3] is vastly greater in size than the iOS reference app catalog [66], which includes only eight apps. All this goes to say that another way our work can be expanded upon is to create a sister project to ours, dedicated to the development of iOS MASTG reference applications. If our goal of creating a set of MASTG reference apps to improve the educational value of the MASTG is further pursued, this would be a natural next step.

9. Conclusion

This thesis examined the state of the art in intentionally vulnerable Android app design, as found in the OWASP MASTG reference apps. Furthermore, it conducted a literature review to understand how the overarching OWASP MAS project is used in the academic world. Building on this knowledge, a set of intentionally vulnerable Android apps was designed and developed. The purpose of this was to expand the MASTG reference app catalog and improve its function as a teaching resource and as a benchmark dataset for different types of mobile app security testing tools.

The literature review showed us the main use-case of the OWASP MAS project in academic works. It consists of using the MASTG to evaluate mobile apps based on the quality of their security and whether they meet the requirements laid out by the MASVS. In particular, the MASTG penetration testing guide is a popular tool for evaluating app security. Multiple papers used the MASVS vulnerability categories to map and classify real-world Android security vulnerabilities. The MASTG reference apps were used for testing and evaluating newly developed tools that deal with vulnerability detection. Overall, the literature review indicates that the academic world uses the OWASP MAS project mainly as a framework and tool for evaluating mobile app security.

The analysis of the existing OWASP MASTG reference apps showed that only a few have documentation on the vulnerabilities they include. Furthermore, only a small portion of the vulnerabilities described by the OWASP MASWE find any representation in the reference apps. Most vulnerability types have no reference implementation in the apps, whereas a small group of them is implemented multiple times in different ways. The ten most frequently implemented vulnerability types make up more than half of all implemented vulnerabilities, despite there being 117 different vulnerability types described by the OWASP MASWE. The same trend can be found when examining the vulnerability categories themselves. A few categories, such as MASVS-PLATFORM or MASVS-STORAGE, have most of their respective vulnerability types implemented at least once in a MASTG reference app. Other categories see very little representation in the reference app catalog. Not one of the nine vulnerabilities described by MASVS-PRIVACY is included in any of the reference apps.

The apps we developed are designed to reflect the natural ordering of vulnerabilities done by the MASWE. Every MASVS category comprises its own app. These categories are, in turn, part of the whole project in the form of modules. For each app, the vulnerabilities that comprise its designated MASVS category are grouped into separate packages, with one package per implemented MASWE vulnerability. This architecture is further extended with a shared library module containing common attributes. Extensive use of custom templates and inheritance allows for a design that can easily be extended with new vulnerabilities and categories while producing minimal scaling costs. The vulnerability selection focused on MASWE vulnerability types that have not been implemented in any of the existing MASTG reference apps. Instead of working on multiple MASVS categories at once, the vulnerability categories were implemented one by one.

We developed three apps called `maswe_storage`, `maswe_crypto`, and `maswe_platform`. Together, they contain 28 vulnerability types. 18 of these types have not been implemented in any of the existing MASTG reference apps. All 6 and all 18 (excluding one that is deprecated) vulnerability types described by MASVS-STORAGE and MASVS-CRYPTO have been implemented. Through the addition of our apps, the total coverage of MASWE vulnerability types has increased from 35.9% to 51.3%. All implemented vulnerabilities are mapped to their MASWE counterparts and have extensive documentation. This documentation contains information on where they can be found, how they can be exploited, and sometimes how they can be fixed. The apps we developed are an important milestone in building a unified, well structured, and thoroughly documented MASTG reference apps benchmark.

A recurring challenge during the development of the apps was the limited documentation by OWASP on the MASWE vulnerabilities. The OWASP MAS project is constantly evolving, as new vulnerabilities become more relevant and older ones become obsolete. This leads to many of the vulnerabilities described by MASWE still being in development and having very little documentation on how they should be implemented and what they should include. This complicated the development process, as the vulnerabilities themselves need to have adequate depth for them to be useful both as an educational resource and as a benchmark for security testing tools.

This thesis can be expanded upon in many different ways. Half of the vulnerability types described by MASWE still have no representation in the MASTG reference apps. Now that the apps are developed, extensive benchmarking of both static and dynamic security testing tools can be carried out in order to better understand the landscape of mobile app security testing tools. A complementary set of iOS MASTG reference apps can be developed, making use of the architecture and design we built to branch out to the iOS side of the OWASP MAS project.

This work greatly expands the coverage of MASWE vulnerability types made by the existing MASTG reference applications catalog. Through this, it contributes to making Android mobile security more accessible and easy to learn. The extensive documentation of the complete sets of implemented MASVS-STORAGE and MASVS-CRYPTO vulnerabilities builds towards a more systematic environment for mobile security research and education. The applications developed serve as the foundation for a complete OWASP MAS benchmark for evaluating mobile application security analysis tools. The scalable and modular architecture on which the apps have been developed enables them to change alongside the Android mobile security landscape and the OWASP MAS project as they continue to evolve, which allows for easy maintenance and continuous growth as a completely open-source benchmark for mobile application security.

List of Figures

List of Tables

Listings

Bibliography

[1] L. Ferrari, F. Pagano, L. Verderame, and A. Merlo, "The owapp benchmark: An owasp-compliant vulnerable android app dataset," in *2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2025, pp. 569–580. [Online]. Available: https://ieeexplore.ieee.org/document/11129474

[2] T. O. Foundation, "Owasp mobile application security reference applications," 09 2025. [Online]. Available: https://mas.owasp.org/MASTG/apps/#

[3] ——, "Owasp mastg reference applications overview," 09 2025, as of commit Nr: ffde0bdd9bdfd375d93ddb433bd3134e3977c64d. [Online]. Available: https://github.com/OWASP/mastg/tree/ffde0bdd9bdfd375d93ddb433bd3134e3977c64d/apps/android

[4] B. Mueller, "Android uncrackable l1," 09 2025, as of commit Nr: ffde0bdd9bdfd375d93ddb433bd3134e3977c64d. [Online]. Available: https://github.com/OWASP/mastg/blob/ffde0bdd9bdfd375d93ddb433bd3134e3977c64d/apps/android/MASTG-APP-0003.md

[5] ——, "Android uncrackable l2," 09 2025, as of commit Nr: ffde0bdd9bdfd375d93ddb433bd3134e3977c64d. [Online]. Available: https://github.com/OWASP/mastg/blob/ffde0bdd9bdfd375d93ddb433bd3134e3977c64d/apps/android/MASTG-APP-0004.md

[6] ——, "Android uncrackable l3," 09 2025, as of commit Nr: ffde0bdd9bdfd375d93ddb433bd3134e3977c64d. [Online]. Available: https://github.com/OWASP/mastg/blob/ffde0bdd9bdfd375d93ddb433bd3134e3977c64d/apps/android/MASTG-APP-0005.md

[7] ——, "Android uncrackable l4," 09 2025, as of commit Nr: ffde0bdd9bdfd375d93ddb433bd3134e3977c64d. [Online]. Available: https://github.com/OWASP/mastg/blob/ffde0bdd9bdfd375d93ddb433bd3134e3977c64d/apps/android/MASTG-APP-0015.md

[8] ——, “Android license validator,” 09 2025, as of commit Nr: ffde0bdd9bdfd375d93ddb433bd3134e3977c64d. [Online]. Available: https://github.com/OWASP/mastg/blob/ffde0bdd9bdfd375d93ddb433bd3134e3977c64d/apps/android/MASTG-APP-0002.md

[9] M. H. Achim D. Brucker, “Dvhma,” 08 2018, as of commit Nr: 30f76ee118f4cc37bdd9dc897f5241d872bbe2b5. [Online]. Available: https://github.com/logicalhacking/DVHMA/tree/30f76ee118f4cc37bdd9dc897f5241d872bbe2b5

[10] K. S. Abhinav Sejpal, “Digitalbank,” 08 2015, as of commit Nr: 9fae450849d221a22cb16d6491ecabad4c97fbd9. [Online]. Available: https://github.com/CyberScions/Digitalbank/tree/9fae450849d221a22cb16d6491ecabad4c97fbd9

[11] CSPF-Founder, “Dodovulnerablebank,” 10 2015, as of commit Nr: 8b9ac17fe4619720650169b86f9bb842aca5976c. [Online]. Available: https://github.com/CSPF-Founder/DodoVulnerableBank/tree/8b9ac17fe4619720650169b86f9bb842aca5976c

[12] A. B. Jeroen Beckers, “disable-flutter-tls-verification,” 05 2025, as of commit Nr: 9decb9e11afa3269095a130de1e270a1f094c08f. [Online]. Available: https://github.com/NVISOsecurity/disable-flutter-tls-verification/tree/9decb9e11afa3269095a130de1e270a1f094c08f

[13] J. Sydseter, “Mastestapp-android-network,” 05 2025, as of commit Nr: 5e5278c82f77ffc4cd5db9efc32d0b98dc0d7915. [Online]. Available: https://github.com/sydseter/MASTestApp-Android-NETWORK/tree/5e5278c82f77ffc4cd5db9efc32d0b98dc0d7915

[14] S. Schleier, “Mastg-android-kotlin-app,” 10 2022, as of commit Nr: 211e1cb27be30c575a36c6ff3bbe6dfb43182f9f. [Online]. Available: https://github.com/OWASP/MASTG-Hacking-Playground/tree/211e1cb27be30c575a36c6ff3bbe6dfb43182f9f

[15] S. Patnayak, “Androgoat,” 04 2020, as of commit Nr: 1f383989bf8435ba8aea245439e1b7f68f2f985d. [Online]. Available: https://github.com/satishpatnayak/AndroGoat/tree/1f383989bf8435ba8aea245439e1b7f68f2f985d

[16] A. Jakhar, “Diva android,” 01 2016, as of commit Nr: 2bb63b531ba66c22563ac0911973389e2a0cb723. [Online]. Available: https://github.com/payatu/diva-android/tree/2bb63b531ba66c22563ac0911973389e2a0cb723

[17] A. S. Dinesh Shetty, “Insecurebankv2,” 11 2019, as of commit Nr: 3c8cc3a3ce45ed56d0f58f236b541b41f992e51f. [Online]. Available: https://github.com/dineshshetty/Android-InsecureBankv2/tree/3c8cc3a3ce45ed56d0f58f236b541b41f992e51f

[18] S. Schleier, “Mastg-hacking-playground,” 10 2022, as of commit Nr: db219a1011e6735cf0c6c08ba929a27ef40e1873. [Online]. Available: https://github.com/OWASP/MASTG-Hacking-Playground/tree/db219a1011e6735cf0c6c08ba929a27ef40e1873

[19] “Ovaa,” 7 2024, as of commit Nr: 4c60d202918e33d3dc9dd7574260f75f761f6e2b. [Online]. Available: https://github.com/oversecured/ovaa/tree/4c60d202918e33d3dc9dd7574260f75f761f6e2b

[20] R. G. Gaurang Bhatnagar, “Insecureshop,” 09 2023, as of commit Nr: 76ed1772a12c61a60f4a685770482e0e1f6cf4a7. [Online]. Available: https://github.com/hax0rgb/InsecureShop/tree/76ed1772a12c61a60f4a685770482e0e1f6cf4a7

[21] F. W. Jonas Mayer, “Finstergram,” 07 2024, as of commit Nr: 11fd8b7e24dada74a751ca112956bb7944a17ca5. [Online]. Available: https://github.com/netlight/finstergram/tree/11fd8b7e24dada74a751ca112956bb7944a17ca5

[22] A. K. Amit Kumar Prajapat, Vedant Wayal, “Android bugbazaar,” 08 2024, as of commit Nr: fef6a899a684245eb70b2ab4be926f62be849ed9. [Online]. Available: https://github.com/payatu/BugBazaar/tree/fef6a899a684245eb70b2ab4be926f62be849ed9

[23] T. O. Foundation, “Mobile application security weakness enumeration (maswe),” 01 2026, accessed: 2026-01. [Online]. Available: https://mas.owasp.org/MASWE/#

[24] ——, “Masvs v2.1.0 release & masvs-privacy,” 01 2024. [Online]. Available: https://mas.owasp.org/news/2024/01/18/masvs-v210-release--masvs-privacy/

[25] M. Saifulhakim and A. Fajar, "Security assessment for digital wallet payment partner applications using the owasp method: A case study in indonesia," *Journal of System and Management Sciences*, vol. 14, pp. 337–357, 02 2024. [Online]. Available: https://www.researchgate.net/publication/378480579_Security_Assessment_for_Digital_Wallet_Payment_Partner_Applications_using_the_OWASP_Method_A_Case_Study_in_Indonesia

[26] C. Anwar, C. Sumerli, N. Rahayu, and K. Kraugusteeliana, "The application of mobile security framework (mobsf) and mobile application security testing guide to ensure the security in mobile commerce applications," *Jurnal Sistim Informasi dan Teknologi*, vol. 5, pp. 97–102, 06 2023. [Online]. Available: https://www.researchgate.net/publication/372132617_The_Application_of_Mobile_Security_Framework_MOBSF_and_Mobile_Application_Security_Testing_Guide_to_Ensure_the_Security_in_Mobile_Commerce_Applications

[27] S. Amir, D. F. Priambodo, A. A. Ajhari, and A. Widyasuri, "Analysis of fraud attacks using android package kit in indonesia," in *2024 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*, 2024, pp. 285–290. [Online]. Available: https://ieeexplore.ieee.org/document/10732435

[28] G. F. M. Karo-Karo, S. Windarta, Amiruddin, and I. R. Hikmah, "Evaluating compliance of of the xyz ministry's android messaging applications with owasp masvs: A comprehensive case study," in *2024 IEEE 2nd International Conference on Electrical Engineering, Computer and Information Technology (ICEECIT)*, 2024, pp. 35–40. [Online]. Available: https://ieeexplore.ieee.org/document/10859915

[29] D. Aprilliansyah, I. Riadi, and Sunardi, "Penetration testing with owasp mobile for android security optimization," *Journal of Cyber Health and Computer*, vol. 1, no. 1, p. 10–14, 12 2023. [Online]. Available: https://www.jurnalsibermu.com/index.php/jochac/article/view/3

[30] T. H. Chiboora, L. Chacha, T. Byagutangaza, and A. Gueye, "Evaluating mobile banking application security posture using the owasp's masvs framework," in *Proceedings of the 6th ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies*, ser. COMPASS '23. New York, NY, USA: Association for Computing Machinery, 08 2023, p. 99–106. [Online]. Available: https://doi.org/10.1145/3588001.3609367

[31] A. P. Tinuoye, "Evaluating penetration testing methodologies for mobile applications: A comparative study of android and ios security," 08 2025, preprint available via SSRN; not peer-reviewed and not associated with a formal conference proceedings. [Online]. Available: https://ssrn.com/abstract=5384360

[32] E. Vasconcelos, M. Delamaro, and S. Souza, "Mutation testing to support the security testing of android applications," in *Anais do IX Simpósio Brasileiro de Testes de Software Sistemático e Automatizado.* Porto Alegre, RS, Brasil: SBC, 09 2024, pp. 29–38. [Online]. Available: https://sol.sbc.org.br/index.php/sast/article/view/30213

[33] O. Mykhaylova, T. Fedynyshyn, A. Datsiuk, B. Fihol, and H. Hulak, "Mobile application as a critical infrastructure cyberattack surface," *Cybersecurity Providing in Information and Telecommunication Systems II 2023*, vol. 3050, pp. 29–43, 12 2023, deposited at Borys Grinchenko Kyiv Metropolitan University Institutional repository. [Online]. Available: https://elibrary.kubg.edu.ua/id/eprint/47127/

[34] L. Stevenson and S. Das, ""your doctor is spying on you": An analysis of data practices in mobile healthcare applications," 12 2025. [Online]. Available: https://arxiv.org/abs/2510.06015

[35] J. Zhu, K. Li, S. Chen, L. Fan, J. Wang, and X. Xie, "A comprehensive study on static application security testing (sast) tools for android," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3385–3402, 12 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10738442

[36] G. W. W. Mukti and R. Roestam, "Enhancing password manager application security by root detection with usability and security evaluation," in *2023 Eighth International Conference on Informatics and Computing (ICIC)*, 2023, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10382115

[37] X. Zhang, H. Ye, Z. Huang, X. Ye, Y. Cao, Y. Zhang, and M. Yang, "Understanding the (in)security of cross-side face verification systems in mobile apps: A system perspective," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 934–950. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10179474

[38] M. Steinböck, J. Troost, W. Van Beijnum, J. Seredynski, H. Bos, M. Lindorfer, and A. Continella, "Sok: Hardening techniques in the mobile ecosystem — are we

there yet?" in *2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P)*, 2025, pp. 789–806. [Online]. Available: https://ieeexplore.ieee.org/document/11129415

[39] G. LLC, "Intents and intent filters," 10 2025, accessed: 2026-01-20. [Online]. Available: https://developer.android.com/guide/components/activities/intro-activities

[40] ——, "Service," 09 2025, accessed: 2026-01-20. [Online]. Available: https://developer.android.com/reference/android/app/Service

[41] ——, "Content provider basics," 10 2025, accessed: 2026-01-20. [Online]. Available: https://developer.android.com/guide/topics/providers/content-provider-basics

[42] ——, "App manifest overview," 10 2025, accessed: 2026-01-20. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro

[43] ——, "Intents and intent filters," 10 2025, accessed: 2026-01-20. [Online]. Available: https://developer.android.com/guide/components/intents-filters

[44] ——, "Uri," 10 2025, accessed: 2026-01-20. [Online]. Available: https://developer.android.com/reference/android/net/Uri

[45] ——, "Sharedpreferences," 10 2025, accessed: 2026-01-20. [Online]. Available: https://developer.android.com/reference/android/content/SharedPreferences

[46] ——, "Android keystore system," 04 2025, accessed: 2026-01-20. [Online]. Available: https://developer.android.com/privacy-and-security/keystore

[47] ——, "Android application sandbox," 12 2025, accessed: 2026-01-26. [Online]. Available: https://source.android.com/docs/security/app-sandbox

[48] T. O. Foundation, "Owasp mobile application security," 01 2026, accessed: 2026-01. [Online]. Available: https://mas.owasp.org/

[49] ——, "Owasp masvs," 01 2026, accessed: 2026-01. [Online]. Available: https://mas.owasp.org/MASVS/

[50] ——, "Owasp mastg," 01 2026, accessed: 2026-01. [Online]. Available: https://mas.owasp.org/MASTG/

[51] D. Kronig, “Reference_application_for_owasp_mastg,” https://github.com/domi-cmd/Reference_Application_for_OWASP_MASTG, 2026, open-source. MIT License. Accessed: 2026-01-24.

[52] G. LLC, “Android 7.0 behavior changes,” 05 2024, accessed: 2026-01-27. [Online]. Available: https://developer.android.com/about/versions/nougat/android-7.0-changes#perm

[53] OWASP Cheat Sheet Series, “Password storage cheat sheet,” Open Web Application Security Project (OWASP), 12 2022, accessed: 2026-01-28. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[54] D. Bleichenbacher, “Chosen ciphertext attacks against protocols based on the rsa encryption standard,” in *Advances in Cryptology — CRYPTO ’98*. Springer, 1998.

[55] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, “Shattered: First practical collision for sha-1,” https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html, 2017, google Online Security Blog.

[56] *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*, National Institute of Standards and Technology Std. NIST SP 800-38A, 2001, available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf.

[57] “Cyclic redundancy check (crc-32),” https://en.wikipedia.org/wiki/Cyclic_redundancy_check, accessed: 2026-01-24.

[58] O. Corporation, *Class Random (Java Platform SE 8)*, Oracle, 2023, https://docs.oracle.com/javase/8/docs/api/java/util/Random.html, Accessed: 2026-01-24.

[59] G. LLC, “Secure clipboard handling,” 09 2024, accessed: 2026-01-24. [Online]. Available: https://developer.android.com/privacy-and-security/risks/secure-clipboard-handling

[60] R. Vimont and Genymobile, “scrcpy: Display and control your android device,” https://github.com/Genymobile/scrcpy, 2018, open-source. Apache License 2.0. Accessed: 2026-01-24.

[61] D. Kronig, "Reference_application_for_owasp_mastg checklist," https://github.com/domi-cmd/Reference_Application_for_OWASP_MASTG/blob/main/MAS_checklist.md, 2026, open-source. Accessed: 2026-01-24.

[62] MobSF Project, "Mobile security framework (mobsf)," https://github.com/MobSF/Mobile-Security-Framework-MobSF, 2026, accessed: 2026-01-23.

[63] Semgrep, Inc., "Semgrep: Static analysis tool," https://semgrep.dev/, 2026, accessed: 2026-01-23.

[64] National Institute of Standards and Technology, "FIPS PUB 186-4: Digital Signature Standard (DSS)," U.S. Department of Commerce, Tech. Rep., Jul. 2013, federal Information Processing Standards Publication. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[65] T. O. Foundation, "Owasp vulnerabilities stati," 10 2025, as of commit Nr: b58b4a48df9e2fbf88b87bac95eed279dd105bec. [Online]. Available: https://github.com/OWASP/maswe/tree/b58b4a48df9e2fbf88b87bac95eed279dd105bec/weaknesses

[66] ——, "Owasp mastg ios reference applications overview," 09 2025, as of commit Nr: 7cbccda6ab43128eb1a994b97c599fbffbfd520e. [Online]. Available: https://github.com/OWASP/mastg/tree/7cbccda6ab43128eb1a994b97c599fbffbfd520e/apps/ios

A. Full MobSF Scan Results

MobSF

ANDROID STATIC ANALYSIS REPORT

app_icon

masvs_storage (1.0)

File Name:	masvs_storage_release.apk
Package Name:	com.dkronig.masvs_storage
Scan Date:	Jan. 26, 2026, 4:12 p.m.
App Security Score:	64/100 (LOW RISK)
Grade:	A

FINDINGS SEVERITY

HIGH	MEDIUM	INFO	SECURE	HOTSPOT
0	4	1	1	0

FILE INFORMATION

File Name: masvs_storage_release.apk
Size: 12.71MB
MD5: 4c373aca361fcec5879235a7e129f990
SHA1: fe8dcf737712e22beccbfacf1b95b2b300a5d7ca
SHA256: 5545775ead37e443c5f07da277a176c44fcd063fb3fd6a9a84b13097d58f0052

APP INFORMATION

App Name: masvs_storage
Package Name: com.dkronig.masvs_storage
Main Activity: com.dkronig.masvs_storage.StorageMenu
Target SDK: 35
Min SDK: 35
Max SDK:
Android Version Name: 1.0
Android Version Code: 1

APP COMPONENTS

Activities: 18
Services: 0
Receivers: 1
Providers: 2
Exported Activities: 0
Exported Services: 0
Exported Receivers: 1
Exported Providers: 0

CERTIFICATE INFORMATION

Binary is signed
v1 signature: False
v2 signature: True
v3 signature: False
v4 signature: False
X.509 Subject: CN=Dominic Kronig, OU=University, O=University of Bern, L=Bern, ST=Bern, C=3012
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2025-12-31 12:08:44+00:00
Valid To: 2050-12-25 12:08:44+00:00
Issuer: CN=Dominic Kronig, OU=University, O=University of Bern, L=Bern, ST=Bern, C=3012
Serial Number: 0x1
Hash Algorithm: sha256
md5: 71d59136f94aa3f5454bae353851cb7b
sha1: 62a94e3eabf027ca5dfdd42d912c2c5be84bd50b
sha256: 4f2afb4af5b8974656740ed44235ef4d3e4cff3b027afdde279058ad2fe13754
sha512: e7ed3abdb5695aed963f9c948b0be09ef3bbf09a1a410cd4b6efcd3e2b1d24e834d73a9642b15db205c81d71e539ea7bb933336d7248426d562089edb8bda536
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: 77428f4f4c526be66d196301e1335f4b576b37d16f9892b220b0ef75c37c346a
Found 1 unique certificates

APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO	DESCRIPTION
com.dkronig.masvs_storage.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION	unknown	Unknown permission	Unknown permission from android reference

APKID ANALYSIS

FILE	DETAILS
classes.dex	

FINDINGS	DETAILS
yara_issue	yara issue - dex file recognized by apkid but not yara module
Anti-VM Code	Build.FINGERPRINT check Build.MANUFACTURER check
Compiler	unknown (please file detection issue!)

FILE	DETAILS
classes2.dex	FINDINGS: yara_issue — DETAILS: yara issue - dex file recognized by apkid but not yara module; FINDINGS: Compiler — DETAILS: unknown (please file detection issue!)

FINDINGS	DETAILS
yara_issue	yara issue - dex file recognized by apkid but not yara module
Compiler	unknown (please file detection issue!)

NETWORK SECURITY

NO	SCOPE	SEVERITY	DESCRIPTION

CERTIFICATE ANALYSIS

HIGH: **0** | WARNING: **0** | INFO: **1**

TITLE	SEVERITY	DESCRIPTION
Signed Application	info	Application is signed with a code signing certificate

MANIFEST ANALYSIS

HIGH: **0** | WARNING: **2** | INFO: **0** | SUPPRESSED: **0**

NO	ISSUE	SEVERITY	DESCRIPTION
1	Application Data can be Backed up [android:allowBackup=true]	warning	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
2	Broadcast Receiver (androidx.profileinstaller.ProfileInstallReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.DUMP [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

CODE ANALYSIS

HIGH: **0** | WARNING: **1** | INFO: **1** | SECURE: **0** | SUPPRESSED: **0**

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	com/dkronig/masvs_storage/maswe_0001/RegisterActivity.java
2	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	CWE: CWE-312: Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14	com/dkronig/common/BaseLoginActivity.java com/dkronig/common/BaseRegisterActivity.java com/dkronig/masvs_storage/maswe_0006/EncryptionHandler.java

NIAP ANALYSIS v1.3

NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION

BEHAVIOUR ANALYSIS

RULE ID	BEHAVIOUR	LABEL	FILES
00125	Check if the given file path exist	file	com/dkronig/common/BaseLoginActivity.java

ABUSED PERMISSIONS

TYPE	MATCHES	PERMISSIONS
Malware Permissions	0/25	
Other Common Permissions	0/44	

Malware Permissions:
Top permissions that are widely abused by known malware.

Other Common Permissions:
Permissions that are commonly abused by known malware.

HARDCODED SECRETS

POSSIBLE SECRETS
39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319
6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151
39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308152294913554433653942643
051953eb9618e1c9a1f929a21a0b68540eea2da725b99b315f3b8b489918ef109e156193951ec7e937b1652c0bd3bb1bf073573df883d2c34f1ef451fd46b503f00
115792089210356248762697446949407573530086143415290314195533631308867097853951
6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296
11839296a789a3bc0045c8a5fb42c7d1bd998f54449579b446817afbd17273e662c97ee72995ef42640c550b9013fad0761353c7086a272c24088be94769fd16650
3617de4a96262c6f5d9e98bf9292dc29f8f41dbd289a147ce9da3113b5f0b8c00a60b1ce1d7e819d7a431d7c90ea0e5f
4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5
c6858e06b70404e9cd9e3ecb662395b4429c648139053fb521f828af606b4d3dbaa14b5e77efe75928fe1dc127a2ffa8de3348b3c1856a429bf97e7e31c2e5bd66
b3312fa7e23ee7e4988e056be3f82d19181d9c6efe8141120314088f5013875ac656398d8a2ed19d2a85c8edd3ec2aef
aa87ca22be8b05378eb1c71ef320ad746e1d3b628ba79b9859f741e082542a385502f25dbf55296c3a545e3872760ab7
115792089210356248762697446949407573529996955224135760342422259061068512044369
5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449

POSSIBLE SECRETS

SCAN LOGS

Timestamp	Event	Error
2026-01-26 16:13:02	Generating Hashes	OK
2026-01-26 16:13:02	Extracting APK	OK
2026-01-26 16:13:02	Unzipping	OK
2026-01-26 16:13:02	Parsing APK with androguard	OK
2026-01-26 16:13:03	Extracting APK features using aapt/aapt2	OK
2026-01-26 16:13:03	Getting Hardcoded Certificates/Keystores	OK
2026-01-26 16:13:10	Parsing AndroidManifest.xml	OK

2026-01-26 16:13:10	Extracting Manifest Data	OK
2026-01-26 16:13:10	Manifest Analysis Started	OK
2026-01-26 16:13:10	Performing Static Analysis on: masvs_storage (com.dkronig.masvs_storage)	OK
2026-01-26 16:13:10	Fetching Details from Play Store: com.dkronig.masvs_storage	OK
2026-01-26 16:13:10	Checking for Malware Permissions	OK
2026-01-26 16:13:10	Fetching icon path	OK
2026-01-26 16:13:10	Library Binary Analysis Started	OK
2026-01-26 16:13:10	Reading Code Signing Certificate	OK
2026-01-26 16:13:11	Running APKiD 3.0.0	OK
2026-01-26 16:13:13	Detecting Trackers	OK
2026-01-26 16:13:16	Decompiling APK to Java with JADX	OK

2026-01-26 16:14:33	Converting DEX to Smali	OK
2026-01-26 16:14:33	Code Analysis Started on - java_source	OK
2026-01-26 16:14:39	Android SBOM Analysis Completed	OK
2026-01-26 16:15:13	Android SAST Completed	OK
2026-01-26 16:15:13	Android API Analysis Started	OK
2026-01-26 16:15:15	Android API Analysis Completed	OK
2026-01-26 16:15:16	Android Permission Mapping Started	OK
2026-01-26 16:15:16	Android Permission Mapping Completed	OK
2026-01-26 16:15:17	Android Behaviour Analysis Started	OK
2026-01-26 16:15:50	Android Behaviour Analysis Completed	OK
2026-01-26 16:15:50	Extracting Emails and URLs from Source Code	OK

2026-01-26 16:15:51	Email and URL Extraction Completed	OK
2026-01-26 16:15:51	Extracting String data from APK	OK
2026-01-26 16:15:51	Extracting String data from Code	OK
2026-01-26 16:15:51	Extracting String values and entropies from Code	OK
2026-01-26 16:15:57	Performing Malware check on extracted domains	OK
2026-01-26 16:15:57	Saving to Database	OK

Report Generated by - MobSF v4.4.5

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

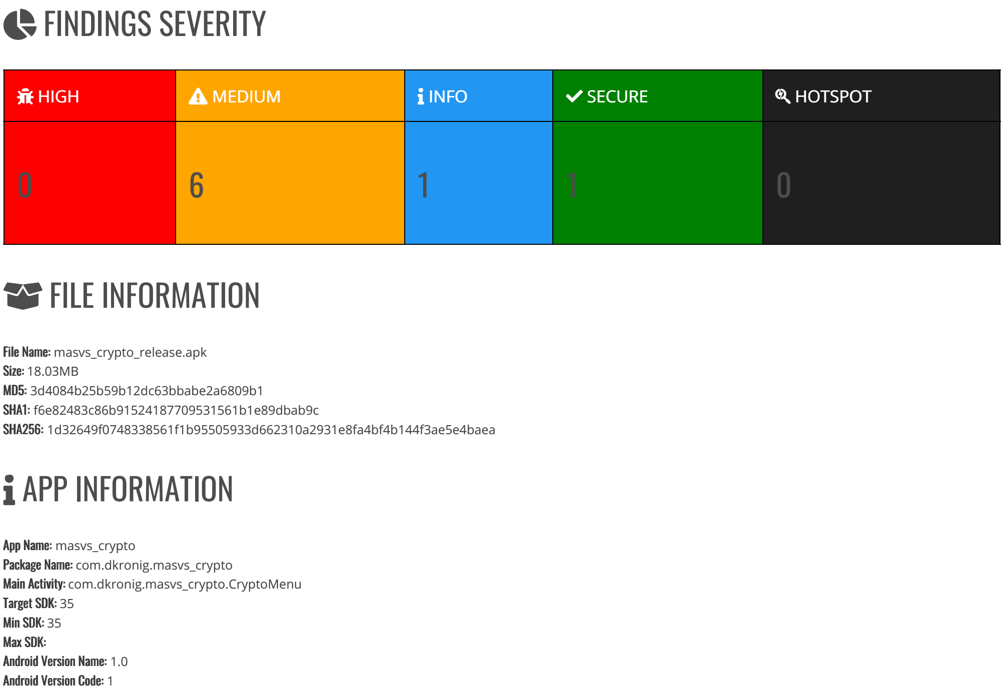
MobSF

ANDROID STATIC ANALYSIS REPORT

app_icon

masvs_crypto (1.0)

File Name:	masvs_crypto_release.apk
Package Name:	com.dkronig.masvs_crypto
Scan Date:	Jan. 26, 2026, 1:45 p.m.
App Security Score:	60/100 (LOW RISK)
Grade:	A



FINDINGS SEVERITY

HIGH	MEDIUM	INFO	SECURE	HOTSPOT
0	6	1	1	0

FILE INFORMATION

File Name: masvs_crypto_release.apk
Size: 18.03MB
MD5: 3d4084b25b59b12dc63bbabe2a6809b1
SHA1: f6e82483c86b915241877095315б1b1e89dbab9c
SHA256: 1d32649f0748338561f1b95505933d662310a2931e8fa4bf4b144f3ae5e4baea

APP INFORMATION

App Name: masvs_crypto
Package Name: com.dkronig.masvs_crypto
Main Activity: com.dkronig.masvs_crypto.CryptoMenu
Target SDK: 35
Min SDK: 35
Max SDK:
Android Version Name: 1.0
Android Version Code: 1

APP COMPONENTS

Activities: 74
Services: 5
Receivers: 1
Providers: 1
Exported Activities: 0
Exported Services: 0
Exported Receivers: 1
Exported Providers: 0

CERTIFICATE INFORMATION

Binary is signed
v1 signature: False
v2 signature: True
v3 signature: False
v4 signature: False
X.509 Subject: CN=Dominic Kronig, OU=University, O=University of Bern, L=Bern, ST=Bern, C=3012
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2025-12-31 12:08:44+00:00
Valid To: 2050-12-25 12:08:44+00:00
Issuer: CN=Dominic Kronig, OU=University, O=University of Bern, L=Bern, ST=Bern, C=3012
Serial Number: 0x1
Hash Algorithm: sha256
md5: 71d59136f94aa3f5454bae353851cb7b
sha1: 62a94e3eabf027ca5dfdd42d912c2c5be84bd50b
sha256: 4f2afb4af5b8974656740ed44235ef4d3e4cff3b027afdde279058ad2fe13754
sha512: e7ed3abdb5695aed963f9c948b0be09ef3bbf09a1a410cd4b6efcd3e2b1d24e834d73a9642b15db205c81d71e539ea7bb933336d7248426d562089edb8bda536
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: 77428f4f4c526be66d196301e1335f4b576b37d16f9892b220b0ef75c37c346a
Found 1 unique certificates

APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO	DESCRIPTION
com.dkronig.masvs_crypto.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION	unknown	Unknown permission	Unknown permission from android reference

APKID ANALYSIS

FILE	FINDINGS	DETAILS
classes.dex	yara_issue	yara issue - dex file recognized by apkid but not yara module
	Anti-VM Code	Build.FINGERPRINT check Build.MANUFACTURER check
	Compiler	unknown (please file detection issue!)

<table>
<tr><th>FILE</th><th>DETAILS</th></tr>
<tr><td>classes2.dex</td><td>
<table>
<tr><th>FINDINGS</th><th>DETAILS</th></tr>
<tr><td>yara_issue</td><td>yara issue - dex file recognized by apkid but not yara module</td></tr>
<tr><td>Compiler</td><td>unknown (please file detection issue!)</td></tr>
</table>
</td></tr>
</table>

NETWORK SECURITY

NO	SCOPE	SEVERITY	DESCRIPTION

CERTIFICATE ANALYSIS

HIGH: **0** | WARNING: **0** | INFO: **1**

TITLE	SEVERITY	DESCRIPTION
Signed Application	info	Application is signed with a code signing certificate

MANIFEST ANALYSIS

HIGH: **0** | WARNING: **2** | INFO: **0** | SUPPRESSED: **0**

NO	ISSUE	SEVERITY	DESCRIPTION
1	Application Data can be Backed up [android:allowBackup] flag is missing.	warning	The flag [android:allowBackup] should be set to false. By default it is set to true and allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
2	Broadcast Receiver (androidx.profileinstaller.ProfileInstallReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.DUMP [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

</> CODE ANALYSIS

HIGH: **0** | WARNING: **3** | INFO: **1** | SECURE: **0** | SUPPRESSED: **0**

NO	ISSUE	SEVERITY	STANDARDS	FILES

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	CWE: CWE-312: Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14	com/dkronig/common/BaseLoginActivity.java com/dkronig/common/BaseRegisterActivity.java com/dkronig/masvs_crypto/maswe_0009/EncryptionHandler.java com/dkronig/masvs_crypto/maswe_0010/EncryptionHandler.java com/dkronig/masvs_crypto/maswe_0011/EncryptionHandler.java com/dkronig/masvs_crypto/maswe_0014/EncryptionHandler.java com/dkronig/masvs_crypto/maswe_0015/EncryptionHandler.java com/dkronig/masvs_crypto/maswe_0022/EncryptionHandler.java com/dkronig/masvs_crypto/maswe_0023/EncryptionHandler.java com/dkronig/masvs_crypto/maswe_0024/EncryptionHandler.java com/dkronig/masvs_crypto/maswe_0027/EncryptionHandler.java
2	SHA-1 is a weak hash known to have hash collisions.	warning	CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-4	com/dkronig/masvs_crypto/maswe_0021/EncryptionHandler.java
3	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	com/dkronig/masvs_crypto/maswe_0023/EncryptionHandler.java
4	The App uses an insecure Random Number Generator.	warning	CWE: CWE-330: Use of Insufficiently Random Values OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-6	com/dkronig/masvs_crypto/maswe_0027/EncryptionHandler.java

NIAP ANALYSIS v1.3

NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION

BEHAVIOUR ANALYSIS

RULE ID	BEHAVIOUR	LABEL	FILES
00125	Check if the given file path exist	file	com/dkronig/common/BaseLoginActivity.java

ABUSED PERMISSIONS

TYPE	MATCHES	PERMISSIONS
Malware Permissions	0/25	
Other Common Permissions	0/44	

Malware Permissions:
Top permissions that are widely abused by known malware.
Other Common Permissions:
Permissions that are commonly abused by known malware.

HARDCODED SECRETS

POSSIBLE SECRETS
E95E4A5F737059DC60DF5991D45029409E60FC09
6A941977BA9F6A435199ACFC51067ED587F519C5ECB541B8E44111DE1D40
0418DE98B02DB9A306F2AFCD7235F72A819B80AB12EBD653172476FECD462AABFFC4FF191B946A5F54D8D0AA2F418808CC25AB056962D30651A114AFD2755AD33 6747F93475B7A1FCA3B88F2B6A208CCFE469408584DC2B2912675BF5B9E582928
B3312FA7E23EE7E4988E056BE3F82D19181D9C6EFE8141120314088F5013875AC656398D8A2ED19D2A85C8EDD3EC2AEF
0100FAF51354E0E39E4892DF6E319C72C8161603FA45AA7B998A167B8F1E629521
FFFFFFFFFFFFFFFFADF85458A2BB4A9AAFDC5620273D3CF1D8B9C583CE2D3695A9E13641146433FBCC939DCE249B3EF97D2FE363630C75D8F681B202AEC4617AD3D F1ED5D5FD65612433F51F5F066ED0856365553DED1AF3B557135E7F57C935984F0C70E0E68B77E2A689DAF3EFE8721DF158A136ADE73530ACCA4F483A797ABC0AB1 82B324FB61D108A94BB2C8E3FBB96ADAB760D7F4681D4F42A3DE394DF4AE56EDE76372BB190B07A7C8EE0A6D709E02FCE1CDF7E2ECC03404CD28342F619172FE9C E98583FF8E4F1232EEF28183C3FE3B1B4C6FAD733BB5FCBC2EC22005C58EF1837D1683B2C6F34A26C1B2EFFA886B4238611FCFDCDE355B3B6519035BBC34F4DEF99 C023861B46FC9D6E6C9077AD91D2691F7F7EE598CB0FAC186D91CAEFE130985139270B4130C93BC437944F4FD4452E2D74DD364F2E21E71F54BFF5CAE82AB9C9DF6 9EE86D2BC522363A0DABC521979B0DEADA1DBF9A42D5C4484E0ABCD06BFA53DDEF3C1B20EE3FD59D7C25E41D2B669E1EF16E6F52C3164DF4FB7930E9E4E58857B 6AC7D5F42D69F6D187763CF1D5503400487F55BA57E31CC7A7135C886EFB4318AED6A1E012D9E6832A907600A918130C46DC778F971AD0038092999A333CB8B7A1 A1DB93D7140003C2A4ECEA9F98D0ACC0A8291CDCEC97DCF8EC9B55A7F88A46B4DB5A851F44182E1C68A007E5E0DD9020BFD64B645036C7A4E677D2C38532A3A23 BA4442CAF53EA63BB454329B7624C8917BDD64B1C0FD4CB38E8C334C701C3ACDAD0657FCCFEC719B1F5C3E4E46041F388147FB4CFDB477A52471F7A9A96910B855 322EDB6340D8A00EF092350511E30ABEC1FFF9E3A26E7FB29F8C183023C3587E38DA0077D9B4763E4E4B94B2BBC194C6651E77CAF992EEAAC0232A281BF6B3A739C 1226116820AE8DB5847A67CBEF9C9091B462D538CD72B03746AE77F5E62292C311562A846505DC82DB854338AE49F5235C95B91178CCF2DD5CACEF403EC9D1810C 6272B045B3B71F9DC6B80D63FDD4A8E9ADB1E6962A69526D43161C1A41D570D7938DAD4A40E329CCFF46AAA36AD004CF600C8381E425A31D951AE64FDB23FCEC 9509D43687FEB69EDD1CC5E0B8CC3BDF64B10EF86B63142A3AB8829555B2F747C932665CB2C0F1CC01BD70229388839D2AF05E454504AC78B7582822846C0BA35C 35F5C59160CC046FD8251541FC68C9C86B022BB7099876A460E7451A8A93109703FEE1C217E6C3826E52C51AA691E0E423CFC99E9E31650C1217B624816CDAD9A95 F9D5B8019488D9C0A0A1FE3075A577E23183F81D4A3F2FA4571EFC8CE0BA8A4FE8B6855DFE72B0A66EDED2FBABFBE58A30FAFABE1C5D71A87E2F741EF8C1FE86FEA 6BBFDE530677F0D97D11D49F7A8443D0822E506A9F4614E011E2A94838FF88CD68C8BB7C5C6424CFFFFFFFFFFFFFFFF
b28ef557ba31dfcbdd21ac46e2a91e3c304f44cb87058ada2cb815151e610046
a335926aa319a27a1d00896a6773a4827acdac73
10686D41FF744D4449FCCF6D8EEA03102E6812C93A9D60B978B702CF156D814EF

POSSIBLE SECRETS
96341f1138933bc2f503fd44
051953eb9618e1c9a1f929a21a0b68540eea2da725b99b315f3b8b489918ef109e156193951ec7e937b1652c0bd3bb1bf073573df883d2c34f1ef451fd46b503f00
040503213F78CA44883F1A3B8162F188E553CD265F23C1567A16876913B0C2AC245849283601CCDA380F1C9E318D90F95D07E5426FE87E45C0E8184698E45962364E34116177DD2259
790408F2EEDAF392B012EDEFB3392F30F4327C0CA3F31FC383C422AA8C16
010092537397ECA4F6145799D62B0A19CE06FE26AD
0443BD7E9AFB53D8B85289BCC48EE5BFE6F20137D10A087EB6E7871E2A10A599C710AF8D0D39E2061114FDD05545EC1CC8AB4093247F77275E0743FFED117182EAA9C77877AAAC6AC7D35245D1692E8EE1
B4E134D3FB59EB8BAB57274904664D5AF50388BA
D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B32E27
11839296a789a3bc0045c8a5fb42c7d1bd998f54449579b446817afbd17273e662c97ee72995ef42640c550b9013fad0761353c7086a272c24088be94769fd16650
41ECE55743711A8C3CBF3783CD08C0EE4D4DC440D4641A8F366E550DFDB3BB67
D35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B32E24
0403F0EBA16286A2D57EA0991168D4994637E8343E3600D51FBC6C71A0094FA2CDD545B11C5C0C797324F1
6db14acc9e21c820ff28b1d5ef5de2b0
22123dc2395a05caa7423daeccc94760a7d462256bd56916
4D696E676875615175985BD3ADBADA21B43A97E2

POSSIBLE SECRETS

F5CE40D95B5EB899ABBCCFF5911CB8577939804D6527378B8C108C3D2090FF9BE18E2D33E3021ED2EF32D85822423B6304F726AA854BAE07D0396E9A9ADDC40F
10D9B4A3D9047D8B154359ABFB1B7F5485B04CEB868237DDC9DEDA982A679A5A919B626D4E50A8DD731B107A9962381FB5D807BF2618
EEAF0AB9ADB38DD69C33F80AFA8FC5E86072618775FF3C0B9EA2314C9C256576D674DF7496EA81D3383B4813D692C6E0E0D5D8E250B98BE48E495C1D6089DAD15 DC7D7B46154D6B6CE8EF4AD69B15D4982559B297BCF1885C529F566660E57EC68EDBC3C05726CC02FD4CBF4976EAA9AFD5138FE8376435B9FC61D2FC0EB06E3
3FCDA526B6CDF83BA1118DF35B3C31761D3545F32728D003EEB25EFE96
f8183668ba5fc5bb06b5981e6d8b795d30b8978d43ca0ec572e37e09939a9773
6b8cf07d4ca75c88957d9d670591
0236B3DAF8A23206F9C4F299D7B21A9C369137F2C84AE1AA0D
0307AF69989546103D79329FCC3D74880F33BBE803CB
5667676A654B20754F356EA92017D946567C46675556F19556A04616B567D223A5E05656FB549016A96656A557
5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B
B99B99B099B323E02709A4D696E6768756151751
E95E4A5F737059DC60DFC7AD95B3D8139515620C
32879423AB1A0375895786C4BB46E9565FDE0B5344766740AF268ADB32322E5C
048BD2AEB9CB7E57CB2C4B482FFC81B7AFB9DE27E1E3BD23C23A4453BD9ACE3262547EF835C3DAC4FD97F8461A14611DC9C27745132DED8E545C1D54C72F04699 7

POSSIBLE SECRETS

3EE30B568FBAB0F883CCEBD46D3F3BB8A2A73513F5EB79DA66190EB085FFA9F492F375A97D860EB4
04B6B3D4C356C139EB31183D4749D423958C27D2DCAF98B70164C97A2DD98F5CFF6142E0F7C8B204911F9271F0F3ECEF8C2701C307E8E4C9E183115A1554062CFB
04A3E8EB3CC1CFE7B7732213B23A656149AFA142C47AAFBC2B79A191562E1305F42D996C823439C56D7F7B22E14644417E69BCB6DE39D027001DABE8F35B25C9BE
7d7374168ffe3471b60a857686a19475d3bfa2ff
5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
7F519EADA7BDA81BD826DBA647910F8C4B9346ED8CCDC64E4B1ABD11756DCE1D2074AA263B88805CED70355A33B471EE
469A28EF7C28CCA3DC721D044F4496BCCA7EF4146FBF25C9
2E45EF571F00786F67B0081B9495A3D95462F5DE0AA185EC
6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296
01AF286BCA1AF286BCA1AF286BCA1AF286BCA1AF286BC9FB8F6B85C556892C20A7EB964FE7719E74F490758D3B
7A1F6653786A68192803910A3D30B2A2018B21CD54
FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551
F1FD178C0B3AD58F10126DE8CE42435B3961ADBCABC8CA6DE8FCF353D86E9C03
00689918DBEC7E5A0DD6DFC0AA55C7

POSSIBLE SECRETS
64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1
07A11B09A76B562144418FF3FF8C2570B8
5F49EB26781C0EC6B8909156D98ED435E45FD59918
00C9BB9E8927D4D64C377E2AB2856A5B16E3EFB7F61D4316AE
8138e8a0fcf3a4e84a771d40fd305d7f4aa59306d7251de54d98af8fe95729a1f73d893fa424cd2edc8636a6c3285e022b0e3866a565ae8108eed8591cd4fe8d2ce86165a9 78d719ebf647f362d33fca29cd179fb42401cbaf3df0c614056f9c8f3cfd51e474afb6bc6974f78db8aba8e9e517fded658591ab7502bd41849462f
9ba48cba5ebcb9b6bd33b92830b2a2e0e192f10a
b3312fa7e23ee7e4988e056be3f82d19181d9c6efe8141120314088f5013875ac656398d8a2ed19d2a85c8edd3ec2aef
02120FC05D3C67A99DE161D2F4092622FECA701BE4F50F4758714E8A87BBF2A658EF8C21E7C5EFE965361F6C2999C0C247B0DBD70CE6B7
24B7B137C8A14D696E6768756151756FD0DA2E5C
D35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E98691555B44C59311
FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1 356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381FFFFFFFFFFFFFFFF
3940200619639447921227904010014361380507973927046544666794690527962765939911326356939895630815229491355443365394264 3
4230017757A767FAE42398569B746325D45313AF0766266479B75654E65F
714114B762F2FF4A7912A6D2AC58B9B5C2FCFE76DAEB7129
0452DCB034293A117E1F4FF11B30F7199D3144CE6DFEAFFEF2E331F296E071FA0DF9982CFEA7D43F2E

POSSIBLE SECRETS

4E13CA542744D696E67687561517552F279A8C84
DB7C2ABF62E35E668076BEAD2088
91771529896554605945588149018382750217296858393520724172743325725474374979801
7ae96a2b657c07106e64479eac3434e99cf0497512f58995c1396c28719501ee
046AB1E344CE25FF3896424E7FFE14762ECB49F8928AC0C76029B4D5800374E9F5143E568CD23F3F4D7C0D4B1E41C8CC0D1C6ABD5F1A46DB4C
115792089210356248762697446949407573530086143415290314195533631308867097853951
3617de4a96262c6f5d9e98bf9292dc29f8f41dbd289a147ce9da3113b5f0b8c00a60b1ce1d7e819d7a431d7c90ea0e5f
D6031998D1B3BBFEBF59CC9BBFF9AEE1
03375D4CE24FDE434489DE8746E71786015009E66E38A926DD
AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F3
fe0e87005b4e83761908c5131d552a850b3f58b749c37cf5b84d6768
103FAEC74D696E676875615175777FC5B191EF30
91E38443A5E82C0D880923425712B2BB658B9196932E02C78B2582FE742DAA28
1A62BA79D98133A16BBAE7ED9A8E03C32E0824D57AEF72F88986874E5AAE49C27BED49A2A95058068426C2171E99FD3B43C5947C857D

POSSIBLE SECRETS
1CEF494720115657E18F938D7A7942394FF9425C1458C57861F9EEA6ADBE3BE10
5363ad4cc05c30e0a5261c028812645a122e22ea20816678df02967c1b23bd72
04161FF7528B899B2D0C28607CA52C5B86CF5AC8395BAFEB13C02DA292DDED7A83
77d0f8c4dad15eb8c4f2f8d6726cefd96d5bb399
00FDFB49BFE6C3A89FACADAA7A1E5BBC7CC1C2E5D831478814
2580F63CCFE44138870713B1A92369E33E2135D266DBB372386C400B
040D9029AD2C7E5CF4340823B2A87DC68C9E4CE3174C1E6EFDEE12C07D58AA56F772C0726F24C6B89E4ECDAC24354B9E99CAA3F6D3761402CD
70B5E1E14031C1F70BBEFE96BDDE66F451754B4CA5F48DA241F331AA396B8D1839A855C1769B1EA14BA53308B5E2723724E090E02DB9
7503CFE87A836AE3A61B8816E25450E6CE5E1C93ACF1ABC1778064FDCBEFA921DF1626BE4FD036E93D75E6A50E3A41E98028FE5FC235F5B889A589CB5215F2A4
7167EFC92BB2E3CE7C8AAAFF34E12A9C557003D7C73A6FAF003F99F6CC8482E540F7
43FC8AD242B0B7A6F3D1627AD5654447556B47BF6AA4A64B0C2AFE42CADAB8F93D92394C79A79755437B56995136
b3fb3400dec5c4adceb8655d4c94
004D696E67687561517512D8F03431FCE63B88F4
1335318132727206734338595199483190012179423759678474868994823595993696425287347124615904033277318214103280125292538719147885989931033105677441361963648030647213778266568986864684632777101508094011826087702016153249904683329312949209127762411378780302243557466062839716593764268326742697808800616315281634758887
DB7C2ABF62E35E668076BEAD208B

POSSIBLE SECRETS

0409487239995A5EE76B55F9C2F098A89CE5AF8724C0A23E0E0FF77500
ffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551
D09E8800291CB85396CC6717393284AAA0DA64BA
3086d221a7d46bcde86c90e49284eb153dab
04925BE9FB01AFC6FB4D3E7D4990010F813408AB106C4F09CB7EE07868CC136FFF3357F624A21BED5263BA3A7A27483EBF6671DBEF7ABB30EBEE084E58A0B077AD42A5A0989D1EE71B1B9BC0455FB0D2C3
6b016c3bdcf18941d0d654921475ca71a9db2fb27d1d37796185c2942c0a
FFFFFFFFFFFFFFFFADF85458A2BB4A9AAFDC5620273D3CF1D8B9C583CE2D3695A9E13641146433FBCC939DCE249B3EF97D2FE363630C75D8F681B202AEC4617AD3DF1ED5D5FD65612433F51F5F066ED0856365553DED1AF3B557135E7F57C935984F0C70E0E68B77E2A689DAF3EFE8721DF158A136ADE73530ACCA4F483A797ABC0AB182B324FB61D108A94BB2C8E3FBB96ADAB760D7F4681D4F42A3DE394DF4AE56EDE76372BB190B07A7C8EE0A6D709E02FCE1CDF7E2ECC03404CD28342F619172FE9CE98583FF8E4F1232EEF28183C3FE3B1B4C6FAD733BB5FCBC2EC22005C58EF1837D1683B2C6F34A26C1B2EFFA886B4238611FCFDCDE355B3B6519035BBC34F4DEF99C023861B46FC9D6E6C9077AD91D2691F7F7EE598CB0FAC186D91CAEFE130985139270B4130C93BC437944F4FD4452E2D74DD364F2E21E71F54BFF5CAE82AB9C9DF69EE86D2BC522363A0DABC521979B0DEADA1DBF9A42D5C4484E0ABCD06BFA53DDEF3C1B20EE3FD59D7C25E41D2B669E1EF16E6F52C3164DF4FB7930E9E4E58857B6AC7D5F42D69F6D187763CF1D5503400487F55BA57E31CC7A7135C886EFB4318AED6A1E012D9E6832A907600A918130C46DC778F971AD0038092999A333CB8B7A1A1DB93D7140003C2A4ECEA9F98D0ACC0A8291CDCEC97DCF8EC9B55A7F88A46B4DB5A851F44182E1C68A007E5E0DD9020BFD64B645036C7A4E677D2C38532A3A23BA4442CAF53EA63BB454329B7624C8917BDD64B1C0FD4CB38E8C334C701C3ACDAD0657FCCFEC719B1F5C3E4E46041F388147FB4CFDB477A52471F7A9A96910B855322EDB6340D8A00EF092350511E30ABEC1FFF9E3A26E7FB29F8C183023C3587E38DA0077D9B4763E4E4B94B2BBC194C6651E77CAF992EEAAC0232A281BF6B3A739C1226116820AE8DB5847A67CBEF9C9091B462D538CD72B03746AE77F5E62292C311562A846505DC82DB854338AE49F5235C95B91178CCF2DD5CACEF403EC9D1810C6272B045B3B71F9DC6B80D63FDD4A8E9ADB1E6962A69526D43161C1A41D570D7938DAD4A40E329CD0E40E65FFFFFFFFFFFFFFFF
6127C24C05F38A0AAAF65C0EF02C
71169be7330b3038edb025f1d0f9

POSSIBLE SECRETS
7B425ED097B425ED097B425ED097B425ED097B425ED097B4260B5E9C7710C864
EE353FCA5428A9300D4ABA754A44C00FDFEC0C9AE4B1A1803075ED967B7BB73F
1f3bdba585295d9a1110d1df1f9430ef8442c5018976ff3437ef91b81dc0b8132c8d5c39c32d0e004a3092b7d327c0e7a4d26d2c7b69b58f9066652911e457779de
7BC86E2102902EC4D5890E8B6B4981ff27E0482750FEFC03
114ca50f7a8e2f3f657c1108d9d44cfd8
04DB4FF10EC057E9AE26B07D0280B7F4341DA5D1B1EAE06C7D9B2F2F6D9C5628A7844163D015BE86344082AA88D95E2F9D
4A6E0856526436F2F88DD07A341E32D04184572BEB710
BB8E5E8FBC115E139FE6A814FE48AAA6F0ADA1AA5DF91985
1A827EF00DD6FC0E234CAF046C6A5D8A85395B236CC4AD2CF32A0CADBDC9DDF620B0EB9906D0957F6C6FEACD615468DF104DE296CD8F
FFFFFFFFFFFFFFFFADF85458A2BB4A9AAFDC5620273D3CF1D8B9C583CE2D3695A9E13641146433FBCC939DCE249B3EF97D2FE363630C75D8F681B202AEC4617AD3D F1ED5D5FD65612433F51F5F066ED0856365553DED1AF3B557135E7F57C935984F0C70E0E68B77E2A689DAF3EFE8721DF158A136ADE73530ACCA4F483A797ABC0AB1 82B324FB61D108A94BB2C8E3FBB96ADAB760D7F4681D4F42A3DE394DF4AE56EDE76372BB190B07A7C8EE0A6D709E02FCE1CDF7E2ECC03404CD28342F619172FE9C E98583FF8E4F1232EEF28183C3FE3B1B4C6FAD733BB5FCBC2EC22005C58EF1837D1683B2C6F34A26C1B2EFFA886B4238611FCFDCDE355B3B6519035BBC34F4DEF99 C023861B46FC9D6E6C9077AD91D2691F7F7EE598CB0FAC186D91CAEFE130985139270B4130C93BC437944F4FD4452E2D74DD364F2E21E71F54BFF5CAE82AB9C9DF6 9EE86D2BC522363A0DABC521979B0DEADA1DBF9A42D5C4484E0ABCD06BFA53DDEF3C1B20EE3FD59D7C25E41D2B66C62E37FFFFFFFFFFFFFFFF
0108B39E77C4B108BED981ED0E890E117C511CF072
04026EB7A859923FBC82189631F8103FE4AC9CA2970012D5D46024804801841CA44370958493B205E647DA304DB4CEB08CBBD1BA39494776FB988B47174DCA88C7 E2945283A01C89720349DC807F4FBF374F4AEADE3BCA95314DD58CEC9F307A54FFC61EFC006D8A2C9D4979C0AC44AEA74FBEBBB9F772AEDCB620B01A7BA7AF1B32 0430C8591984F601CD4C143EF1C7A3

POSSIBLE SECRETS
0340340340340340340340340340340340340340340340340340340323C313FAB50589703B5EC68D3587FEC60D161CC149C1AD4A91
0289FDFBE4ABE193DF9559ECF07AC0CE78554E2784EB8C1ED1A57A
044AD5F7048DE709AD51236DE65E4D4B482C836DC6E410664002BB3A02D4AAADACAE24817A4CA3A1B014B5270432DB27D2
5E5CBA992E0A680D885EB903AEA78E4A45A469103D448EDE3B7ACCC54D521E37F84A4BDD5B06B0970CC2D2BBB715F7B82846F9A0C393914C792E6A923E2117AB8 05276A975AADB5261D91673EA9AAFFEECBFA6183DFCB5D3B7332AA19275AFA1F8EC0B60FB6F66CC23AE4870791D5982AAD1AA9485FD8F4A60126FEB2CF05DB8A7F 0F09B3397F3937F2E90B9E5B9C9B6EFEF642BC48351C46FB171B9BFA9EF17A961CE96C7E7A7CC3D3D03DFAD1078BA21DA425198F07D2481622BCE45969D9C4D606 3D72AB7A0F08B2F49A7CC6AF335E08C4720E31476B67299E231F8BD90B39AC3AE3BE0C6B6CACEF8289A2E2873D58E51E029CAFBD55E6841489AB66B5B4B9BA6E2F 784660896AFF387D92844CCB8B69475496DE19DA2E58259B090489AC8E62363CDF82CFD8EF2A427ABCD65750B506F56DDE3B988567A88126B914D7828E2B63A6D7 ED0747EC59E0E0A23CE7D8A74C1D2C2A7AFB6A29799620F00E11C33787F7DED3B30E1A22D09F1FBDA1ABBBFBF25CAE05A13F812E34563F99410E73B
0202F9F87B7C574D0BDECF8A22E6524775F98CDEBDCB
12511cfe811d0f4e6bc688b4d
FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1 356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA483 61C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C32905E462E36CE3BE39E772C18 0E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497CEA956AE515D2261898FA051015728E5A8AAAC42DAD33170D04507A33A85521ABDF1C BA64ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6BF12FFA06D98A0864D87602733EC86 A64521F2B18177B200CBBE117577A615D6C770988C0BAD946E208E24FA074E5AB3143DB5BFCE0FD108E4B82D120A92108011A723C12A787E6D788719A10BDBA5B2 699C327186AF4E23C1A946834B6150BDA2583E9CA2AD44CE8DBBBC2DB04DE8EF92E8EFC141FBECAA6287C59474E6BC05D99B2964FA090C3A2233BA186515BE7ED 1F612970CEE2D7AFB81BDD762170481CD0069127D5B05AA993B4EA988D8FDDC186FFB7DC90A6C08F4DF435C93402849236C3FAB4D27C7026C1D4DCB2602646DE C9751E763DBA37BDF8FF9406AD9E530EE5DB382F413001AEB06A53ED9027D831179727B0865A8918DA3EDBEBCF9B14ED44CE6CBACED4BB1BDB7F1447E6CC254B3 32051512BD7AF426FB8F401378CD2BF5983CA01C64B92ECF032EA15D1721D03F482D7CE6E74FEF6D55E702F46980C82B5A84031900B1C9E59E7C97FBEC7E8F323A9 7A7E36CC88BE0F1D45B7FF585AC54BD407B22B4154AACC8F6D7EBF48E1D814CC5ED20F8037E0A79715EEF29BE32806A1D58BB7C5DA76F550AA3D8A1FBFF0EB19CC B1A313D55CDA56C9EC2EF29632387FE8D76E3C0468043E8F663F4860EE12BF2D5B0B7474D6E694F91E6DBE115974A3926F12FEE5E438777CB6A932DF8CD8BEC4D07 3B931BA3BC832B68D9DD300741FA7BF8AFC47ED2576F6936BA424663AAB639C5AE4F5683423B4742BF1C978238F16CBE39D652DE3FDB8BEFC848AD922222E04A40 37C0713EB57A81A23F0C73473FC646CEA306B4BCBC8862F8385DDFA9D4B7FA2C087E879683303ED5BDD3A062B3CF5B3A278A66D2A13F83F44F82DDF310EE074AB6 A364597E899A0255DC164F31CC50846851DF9AB48195DED7EA1B1D510BD7EE74D73FAF36BC31ECFA268359046F4EB879F924009438B481C6CD7889A002ED5EE382 BC9190DA6FC026E479558E4475677E9AA9E3050E2765694DFC81F56E880B96E7160C980DD98EDD3DFFFFFFFFFFFFFFFF

POSSIBLE SECRETS
687D1B459DC841457E3E06CF6F5E2517B97C7D614AF138BCBF85DC806C4B289F3E965D2DB1416D217F8B276FAD1AB69C50F78BEE1FA3106EFB8CCBC7C5140116
040369979697AB43897789566789567F787A7876A65400435EDB42EFAFB2989D51FEFCE3C80988F41FF883
e9e642599d355f37c97ffd3567120b8e25c9cd43e927b3a9670fbec5d890141922d2c3b3ad2480093799869d1e846aab49fab0ad26d2ce6a22219d470bce7d777d4a21fbe 9c270b57f607002f3cef8393694cf45ee3688c11a8c56ab127a3daf
4B337D934104CD7BEF271BF60CED1ED20DA14C08B3BB64F18A60888D
0051953EB9618E1C9A1F929A21A0B68540EEA2DA725B99B315F3B8B489918EF109E156193951EC7E937B1652C0BD3BB1BF073573DF883D2C34F1EF451FD46B503F0 0
FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1 356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA483 61C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C32905E462E36CE3BE39E772C18 0E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497CEA956AE515D2261898FA051015728E5A8AAAC42DAD33170D04507A33A85521ABDF1C BA64ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6BF12FFA06D98A0864D87602733EC86 A64521F2B18177B200CBBE117577A615D6C770988C0BAD946E208E24FA074E5AB3143DB5BFCE0FD108E4B82D120A93AD2CAFFFFFFFFFFFFFFFF
8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF3AB6AF6B7FC3103B883202E9046565
072546B5435234A422E0789675F432C89435DE5242
77E2B07370EB0F832A6DD5B62DFC88CD06BB84BE
BDDB97E555A50A908E43B01C798EA5DAA6788F1EA2794EFCF57166B8C14039601E55827340BE

POSSIBLE SECRETS
90066455B5CFC38F9CAA4A48B4281F292C260FEEF01FD61037E56258A7795A1C7AD46076982CE6BB956936C6AB4DCFE05E6784586940CA544B9B2140E1EB523F009D20A7E7880E4E5BFA690F1B9004A27811CD9904AF70420EEFD6EA11EF7DA129F58835FF56B89FAA637BC9AC2EFAAB903402229F491D8D3485261CD068699B6BA58A1DDBBEF6DB51E8FE34E8A78E542D7BA351C21EA8D8F1D29F5D5D15939487E27F4416B0CA632C59EFD1B1EB66511A5A0FBF615B766C5862D0BD8A3FE7A0E0DA0FB2FE1FCB19E8F9996A8EA0FCCDE538175238FC8B0EE6F29AF7F642773EBE8CD5402415A01451A840476B2FCEB0E388D30D4B376C37FE401C2A2C2F941DAD179C540C1C8CE030D460C4D983BE9AB0B20F69144C1AE13F9383EA1C08504FB0BF321503EFE43488310DD8DC77EC5B8349B8BFE97C2C560EA878DE87C11E3D597F1FEA742D73EEC7F37BE43949EF1A0D15C3F3E3FC0A8335617055AC91328EC22B50FC15B941D3D1624CD88BC25F3E941FDDC6200689581BFEC416B4B2CB73
044A96B5688EF573284664698968C38BB913CBFC8223A628553168947D59DCC912042351377AC5FB32
7BC382C63D8C150C3C72080ACE05AFA0C2BEA28E4FB22787139165EFBA91F90F8AA5814A503AD4EB04A8C7DD22CE2826
0021A5C2C8EE9FEB5C4B9A753B7B476B7FD6422EF1F3DD674761FA99D6AC27C8A9A197B272822F6CD57A55AA4F50AE317B13545F
32010857077C5431123A46B808906756F543423E8D27877578125778AC76
002757A1114D696E6768756151755316C05E0BD4
10B51CC12849B234C75E6DD2028BF7FF5C1CE0D991A1
040303001D34B856296C16C0D40D3CD7750A93D1D2955FA80AA5F40FC8DB7B2ABDBDE53950F4C0D293CDD711A35B67FB1499AE60038614F1394ABFA3B4C850D927E1E7769C8EEC2D19037BF27342DA639B6DCCFFFEB73D69D78C6C27A6009CBBCA1980F8533921E8A684423E43BAB08A576291AF8F461BB2A8B3531D2F0485C19B16E2F1516E23DD3C1A4827AF1B8AC15B
91A091F03B5FBA4AB2CCF49C4EDD220FB028712D42BE752B2C40094DBACDB586FB20
71169be7330b3038edb025f1
CFA0478A54717B08CE64805B76E5B14249A77A4838469DF7F7DC987EFCCFB11D
026108BABB2CEEBCF787058A056CBE0CFE622D7723A289E08A07AE13EF0D10D171DD8D

POSSIBLE SECRETS
68A5E62CA9CE6C1C299803A6C1530B514E182AD8B0042A59CAD29F43
28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93
DB7C2ABF62E35E7628DFAC6561C5
030024266E4EB5106D0A964D92C4860E2671DB9B6CC5
8D91E471E0989CDA27DF505A453F2B7635294F2DDF23E3B122ACC99C9E9F1E14
E0D2EE25095206F5E2A4F9ED229F1F256E79A0E2B455970D8D0D865BD94778C576D62F0AB7519CCD2A1A906AE30D
027B680AC8B8596DA5A4AF8A19A0303FCA97FD7645309FA2A581485AF6263E313B79A2F5
04B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6115C1D21BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34
31a92ee2029fd10d901b113e990710f0d21ac6b6
A335926AA319A27A1D00896A6773A4827ACDAC73
3E1AF419A269A5F866A7D3C25C3DF80AE979259373FF2B182F49D4CE7E1BBC8B
c97445f45cdef9f0d3e05e1e585fc297235b82b5be8ff3efca67c59852018192
A7F561E038EB1ED560B3D147DB782013064C19F27ED27C6780AAF77FB8A547CEB5B4FEF422340353
3045AE6FC8422f64ED579528D38120EAE12196D5
9B9F605F5A858107AB1EC85E6B41C8AACF846E86789051D37998F7B9022D7598

POSSIBLE SECRETS
FD0D693149A118F651E6DCE6802085377E5F882D1B510B44160074C1288078365A0396C8E681
043B4C382CE37AA192A4019E763036F4F5DD4D7EBB938CF935318FDCED6BC28286531733C3F03C4FEE
9B9F605F5A858107AB1EC85E6B41C8AACF846E86789051D37998F7B9022D759B
100997906755055304772081815535925224869841082572053457874823515875577147990529272777244152852699298796483356699682842027972896052747173175480590485607134746852141928680912561502802222185647539190902656116367847270145019066794290930185446216399730872221732889830323194097355403213400972588322876850946740663962
8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901D1A71874700133107EC50
041D1C64F068CF45FFA2A63A81B7C13F6B8847A3E77EF14FE3DB7FCAFE0CBD10E8E826E03436D646AAEF87B2E247D4AF1E8ABE1D7520F9C2A45CB1EB8E95CFD55262B70B29FEEC5864E19C054FF99129280E4646217791811142820341263C5315
0405F939258DB7DD90E1934F8C70B0DFEC2EED25B8557EAC9C80E2E198F8CDBECD86B1205303676854FE24141CB98FE6D4B20D02B4516FF702350EDDB0826779C813F0DF45BE8112F4
04AA87CA22BE8B05378EB1C71EF320AD746E1D3B628BA79B9859F741E082542A385502F25DBF55296C3A545E3872760AB73617DE4A96262C6F5D9E98BF9292DC29F8F41DBD289A147CE9DA3113B5F0B8C00A60B1CE1D7E819D7A431D7C90EA0E5F
E87579C11079F43DD824993C2CEE5ED3
0400C6858E06B70404E9CD9E3ECB662395B4429C648139053FB521F828AF606B4D3DBAA14B5E77EFE75928FE1DC127A2FFA8DE3348B3C1856A429BF97E7E31C2E5BD66011839296A789A3BC0045C8A5FB42C7D1BD998F54449579B446817AFBD17273E662C97EE72995EF42640C550B9013FAD0761353C7086A272C24088BE94769FD16650
040081BAF91FDF9833C40F9C181343638399078C6E7EA38C001F73C8134B1B4EF9E150
4D41A619BCC6EADF0448FA22FAD567A9181D37389CA

POSSIBLE SECRETS
0432C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7BC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0
A9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7
1243ae1b4d71613bc9f780a03690e
d09e8800291cb85396cc6717393284aaa0da64ba
04A1455B334DF099DF30FC28A169A467E9E47075A90F7E650EB6B7A45C7E089FED7FBA344282CAFBD6F7E319F7C0B0BD59E2CA4BDB556D61A5
04188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF101207192B95FFC8DA78631011ED6B24CDD573F977A11E794811
07B6882CAAEFA84F9554FF8428BD88E246D2782AE2
023b1660dd701d0839fd45eec36f9ee7b32e13b315dc02610aa1b636e346df671f790f84c5e09b05674dbb7e45c803dd
db92371d2126e9700324977504e8c90e
0095E9A9EC9B297BD4BF36E059184F
3045AE6FC8422F64ED579528D38120EAE12196D5

POSSIBLE SECRETS
FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1 356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA483 61C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C32905E462E36CE3BE39E772C18 0E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497CEA956AE515D2261898FA051015728E5A8AAAC42DAD33170D04507A33A85521ABDF1C BA64ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6BF12FFA06D98A0864D87602733EC86 A64521F2B18177B200CBBE117577A615D6C770988C0BAD946E208E24FA074E5AB3143DB5BFCE0FD108E4B82D120A92108011A723C12A787E6D788719A10BDBA5B2 699C327186AF4E23C1A946834B6150BDA2583E9CA2AD44CE8DBBBC2DB04DE8EF92E8EFC141FBECAA6287C59474E6BC05D99B2964FA090C3A2233BA186515BE7ED 1F612970CEE2D7AFB81BDD762170481CD0069127D5B05AA993B4EA988D8FDDC186FFB7DC90A6C08F4DF435C93402849236C3FAB4D27C7026C1D4DCB2602646DE C9751E763DBA37BDF8FF9406AD9E530EE5DB382F413001AEB06A53ED9027D831179727B0865A8918DA3EDBEBCF9B14ED44CE6CBACED4BB1BDB7F1447E6CC254B3 32051512BD7AF426FB8F401378CD2BF5983CA01C64B92ECF032EA15D1721D03F482D7CE6E74FEF6D55E702F46980C82B5A84031900B1C9E59E7C97FBEC7E8F323A9 7A7E36CC88BE0F1D45B7FF585AC54BD407B22B4154AACC8F6D7EBF48E1D814CC5ED20F8037E0A79715EEF29BE32806A1D58BB7C5DA76F550AA3D8A1FBFF0EB19CC B1A313D55CDA56C9EC2EF29632387FE8D76E3C0468043E8F663F4860EE12BF2D5B0B7474D6E694F91E6DCC4024FFFFFFFFFFFFFFFF
7830A3318B603B89E2327145AC234CC594CBDD8D3DF91610A83441CAEA9863BC2DED5D5AA8253AA10A2EF1C98B9AC8B57F1117A72BF2C7B9E7C1AC4D77FC94CA
b0b4417601b59cbc9d8ac8f935cadaec4f5fbb2f23785609ae466748d9b5a536
216EE8B189D291A0224984C1E92F1D16BF75CCD825A087A239B276D3167743C52C02D6E7232AA
f7e1a085d69b3ddecbbcab5c36b857b97994afbbfa3aea82f9574c0b3d0782675159578ebad4594fe67107108180b449167123e84c281613b7cf09328cc8a6e13c167a8b5 47c8d28e0a3ae1e2bb3a675916ea37f0bfa213562f1fb627a01243bcca4f1bea8519089a883dfe15ae59f06928b665e807b552564014c3bfecf492a
29C41E568B77C617EFE5902F11DB96FA9613CD8D03DB08DA
401028774D7777C7B7666D1366EA432071274F89FF01E718
FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1 356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA483 61C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA237327FFFFFFFFFFFFFFFF
1E589A8595423412134FAA2DBDEC95C8D8675E58
5D9306BACD22B7FAEB09D2E049C6E2866C5D1677762A8F2F2DC9A11C7F7BE8340AB2237C7F2A0

POSSIBLE SECRETS
7988514166341097689762711893575632374730795191650763975830047269233887353 3959
BDB6F4FE3E8B1D9E0DA8C0D40FC962195DFAE76F56564677
E2E31EDFC23DE7BDEBE241CE593EF5DE2295B7A9CBAEF021D385F7074CEA043AA27272A7AE602BF2A7B9033DB9ED3610C6FB85487EAE97AAC5BC7928C1950148
FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C32905E462E36CE3BE39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497CEA956AE515D2261898FA051015728E5A8AACAA68FFFFFFFFFFFFFFFF
6BA06FE51464B2BD26DC57F48819BA9954667022C7D03
659EF8BA043916EEDE8911702B22
A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377
2472E2D0197C49363F1FE7F5B6DB075D52B6947D135D8CA445805D39BC345626089687742B6329E70680231988
85E25BFE5C86226CDB12016F7553F9D0E693A268
0620048D28BCBD03B6249C99182B7C8CD19700C362C46A01
883423532389192164791648750360308885314476597252960362792450860609699839
06973B15095675534C7CF7E64A21BD54EF5DD3B8A0326AA936ECE454D2C
26DC5C6CE94A4B44F330B5D9BBD77CBF958416295CF7E1CE6BCCDC18FF8C07B6
aa87ca22be8b05378eb1c71ef320ad746e1d3b628ba79b9859f741e082542a385502f25dbf55296c3a545e3872760ab7

POSSIBLE SECRETS
6A91174076B1E0E19C39C031FE8685C1CAE040E5C69A28EF
0066647EDE6C332C7F8C0923BB58213B333B20E9CE4281FE115F7D8F90AD
FC1217D4320A90452C760A58EDCD30C8DD069B3C34453837A34ED50CB54917E1C2112D84D164F444F8F74786046A
6b8cf07d4ca75c88957d9d67059037a4
E95E4A5F737059DC60DFC7AD95B3D8139515620F
985BD3ADBAD4D696E676875615175A21B43A97E3
0400FAC9DFCBAC8313BB2139F1BB755FEF65BC391F8B36F8F8EB7371FD558B01006A08A41903350678E58528BEBF8A0BEFF867A7CA36716F7E01F81052
C8619ED45A62E6212E1160349E2BFA844439FAFC2A3FD1638F9E
C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297
3DF91610A83441CAEA9863BC2DED5D5AA8253AA10A2EF1C98B9AC8B57F1117A72BF2C7B9E7C1AC4D77FC94CADC083E67984050B75EBAE5DD2809BD638016F723
c6858e06b70404e9cd9e3ecb662395b4429c648139053fb521f828af606b4d3dbaa14b5e77efe75928fe1dc127a2ffa8de3348b3c1856a429bf97e7e31c2e5bd66
2AA058F73A0E33AB486B0F610410C53A7F132310
39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319
0101D556572AABAC800101D556572AABAC8001022D5C91DD173F8FB561DA6899164443051D
B4C4EE28CEBC6C2C8AC12952CF37F16AC7EFB6A9F69F4B57FFDA2E4F0DE5ADE038CBC2FFF719D2C18DE0284B8BFEF3B52B8CC7A5F5BF0A3C8D2319A5312557E1

POSSIBLE SECRETS
003088250CA6E7C7FE649CE85820F7
0401F481BC5F0FF84A74AD6CDF6FDEF4BF6179625372D8C0C5E10025E399F2903712CCF3EA9E3A1AD17FB0B3201B6AF7CE1B05
0429A0B6A887A983E9730988A68727A8B2D126C44CC2CC7B2A6555193035DC76310804F12E549BDB011C103089E73510ACB275FC312A5DC6B76553F0CA
255705fa2a306654b1f4cb03d6a750a30c250102d4988717d9ba15ab6d3e
0402FE13C0537BBC11ACAA07D793DE4E6D5E5C94EEE80289070FB05D38FF58321F2E800536D538CCDAA3D9
1A8F7EDA389B094C2C071E3647A8940F3C123B697578C213BE6DD9E6C8EC7335DCB228FD1EDF4A39152CBCAAF8C0398828041055F94CEEEC7E21340780FE41BD
021085E2755381DCCCE3C1557AFA10C2F0C0C2825646C5B34A394CBCFA8BC16B22E7E789E927BE216F02E1FB136A5F
00C9517D06D5240D3CFF38C74B20B6CD4D6F9DD4D9
7CBBBCF9441CFAB76E1890E46884EAE321F70C0BCB4981527897504BEC3E36A62BCDFA2304976540F6450085F2DAE145C22553B465763689180EA2571867423E
FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1 356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA483 61C55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746C08CA18217C32905E462E36CE3BE39E772C18 0E86039B2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497CEA956AE515D2261898FA051015728E5A8AAAC42DAD33170D04507A33A85521ABDF1C BA64ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6BF12FFA06D98A0864D87602733EC86 A64521F2B18177B200CBBE117577A615D6C770988C0BAD946E208E24FA074E5AB3143DB5BFCE0FD108E4B82D120A92108011A723C12A787E6D788719A10BDBA5B2 699C327186AF4E23C1A946834B6150BDA2583E9CA2AD44CE8DBBBC2DB04DE8EF92E8EFC141FBECAA6287C59474E6BC05D99B2964FA090C3A2233BA186515BE7ED 1F612970CEE2D7AFB81BDD762170481CD0069127D5B05AA993B4EA988D8FDDC186FFB7DC90A6C08F4DF435C934063199FFFFFFFFFFFFFFFF
BDB6F4FE3E8B1D9E0DA8C0D46F4C318CEFE4AFE3B6B8551F
108576C80499DB2FC16EDDF6853BBB278F6B6FB437D9

POSSIBLE SECRETS
000E0D4D696E6768756151750CC03A4473D03679
5DDA470ABE6414DE8EC133AE28E9BBD7FCEC0AE0FFF2
10E723AB14D696E6768756151756FEBF8FCB49A9
020A601907B8C953CA1481EB10512F78744A3205FD
64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1
bb85691939b869c1d087f601554b96b80cb4f55b35f433c2
03eea2bae7e1497842f2de7769cfe9c989c072ad696f48034a
25FBC363582DCEC065080CA8287AAFF09788A66DC3A9E
4099B5A457F9D69F79213D094C4BCD4D4262210B
8d5155894229d5e689ee01e6018a237e2cae64cd
cc22d6dfb95c6b25e49c0d6364a4e5980c393aa21668d953
8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901D1A71874700133107EC53
678471b27a9cf44ee91a49c5147db1a9aaf244f05a434d6486931d2d14271b9e35030b71fd73da179069b32e2935630e1c2062354d0da20a6c416e50be794ca4
393C7F7D53666B5054B5E6C6D3DE94F4296C0C599E2E2E241050DF18B6090BDC90186904968BB
0163F35A5137C2CE3EA6ED8667190B0BC43ECD69977702709B

POSSIBLE SECRETS
03E5A88919D7CAFCBF415F07C2176573B2
1b9fa3e518d683c6b65763694ac8efbaec6fab44f2276171a42726507dd08add4c3b3f4c1ebc5b1222ddba077f722943b24c3edfa0f85fe24d0c8c01591f0be6f63
04640ECE5C12788717B9C1BA06CBC2A6FEBA85842458C56DDE9DB1758D39C0313D82BA51735CDB3EA499AA77A7D6943A64F7A3F25FE26F06B51BAA2696FA9035DA5B534BD595F5AF0FA2C892376C84ACE1BB4E3019B71634C01131159CAE03CEE9D9932184BEEF216BD71DF2DADF86A627306ECFF96DBB8BACE198B61E00F8B332
D2C0FB15760860DEF1EEF4D696E6768756151754
686479766013060971498190079908139321726943530014330540939446345918554318339765539424505774633321719753296399637136332111386476861244038034037280889270700 5449
02A29EF207D0E9B6C55CD260B306C7E007AC491CA1B10C62334A9E8DCD8D20FB7
e43bb460f0b80cc0c0b075798e948060f8321b7d
0017858FEB7A98975169E171F77B4087DE098AC8A911DF7B01
429418261486158041438734477379555023926723459686071430667981129940894712314200270603852166995638487199576572848148989097707594626134376694563648827303708389347910808359326479767786019153434744009610342313166725786869204821949328786333602033847970926843422476210557602350161326147806527610285094454033386523 41
fd7f53811d75122952df4a9c2eece4e7f611b7523cef4400c31e3f80b6512669455d402251fb593d8d58fabfc5f5ba30f6cb9b556cd7813b801d346ff26660b76b9950a5a49f9fe8047b1022c24fbba9d7feb7c61bf83b57e7c6a8a6150f04fb83f6d3c51ec3023554135a169132f675f3ae2b61d72aeff22203199dd14801c7
020ffa963cdca8816ccc33b8642bedf905c3d358573d3f27fbbd3b3cb9aaaf
0217C05610884B63B9C6C7291678F9D341

POSSIBLE SECRETS
C196BA05AC29E1F9C3C72D56DFFC6154A033F1477AC88EC37F09BE6C5BB95F51C296DD20D1A28A067CCC4D4316A4BD1DCA55ED1066D438C35AEBAABF57E7DAE428782A95ECA1C143DB701FD48533A3C18F0FE23557EA7AE619ECACC7E0B51652A8776D02A425567DED36EABD90CA33A1E8D988F0BBB92D02D1D20290113BB562CE1FC856EEB7CDD92D33EEA6F410859B179E7E789A8F75F645FAE2E136D252BFFAFF89528945C1ABE705A38DBC2D364AADE99BE0D0AAD82E5320121496DC65B3930E38047294FF877831A16D5228418DE8AB275D7D75651CEFED65F78AFC3EA7FE4D79B35F62A0402A1117599ADAC7B269A59F353CF450E6982D3B1702D9CA83
0481AEE4BDD82ED9645A21322E9C4C6A9385ED9F70B5D916C1B43B62EEF4D0098EFF3B1F78E2D0D48D50D1687B93B97D5F7C6D5047406A5E688B352209BCB9F8227DDE385D566332ECC0EABFA9CF7822FDF209F70024A57B1AA000C55B881F8111B2DCDE494A5F485E5BCA4BD88A2763AED1CA2B2FA8F0540678CD1E0F3AD80892
36DF0AAFD8B8D7597CA10520D04B
60dcd2104c4cbc0be6eeefc2bdd610739ec34e317f9b33046c9e4788
04BED5AF16EA3F6A4F62938C4631EB5AF7BDBCDBC31667CB477A1A8EC338F94741669C976316DA6321
340E7BE2A280EB74E2BE61BADA745D97E8F7C300
03188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012
046B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C2964FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5
520883949DFDBC42D3AD198640688A6FE13F41349554B49ACC31DCCD884539816F5EB4AC8FB1F1A6
7A556B6DAE535B7B51ED2C4D7DAA7A0B5C55F380
42debb9da5b3d88cc956e08787ec3f3a09bba5f48b889a74aaf53174aa0fbe7e3c5b8fcd7a53bef563b0e98560328960a9517f4014d3325fc7962bf1e049370d76d1314a76137e792f3f0db859d095e4a5b932024f079ecf2ef09c797452b0770e1350782ed57ddf794979dcef23cb96f183061965c4ebc93c9c71c56b925955a75f94cccf1449ac43d586d0beee43251b0b2287349d68de0d144403f13e802f4146d882e057af19b6f6275c6676c8fa0e3ca2713a3257fd1b27d0639f695e347d8d1cf9ac819a26ca9b04cb0eb9b7b035988d15bbac65212a55239cfc7e58fae38d7250ab9991ffbc97134025fe8ce04c4399ad96569be91a546f4978693c7a
MQVwithSHA384KDFAndSharedInfo

POSSIBLE SECRETS
127971af8721782ecffa3
0667ACEB38AF4E488C407433FFAE4F1C811638DF20
0228F9D04E900069C8DC47A08534FE76D2B900B7D7EF31F5709F200C4CA205
07A526C63D3E25A256A007699F5447E32AE456B50E
02197B07845E9BE2D96ADB0F5F3C7F2CFFBD7A3EB8B6FEC35C7FD67F26DDF6285A644F740A2614
C302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1
C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86294
0238af09d98727705120c921bb5e9e26296a3cdcf2f35757a0eafd87b830e7
295F9BAE7428ED9CCC20E7C359A9D41A22FCCD9108E17BF7BA9337A6F8AE9513
04015D4860D088DDB3496B0C6064756260441CDE4AF1771D4DB01FFE5B34E59703DC255A868A1180515603AEAB60794E54BB7996A70061B1CFAB6BE5F32BBFA783 24ED106A7636B9C5A7BD198D0158AA4F5488D08F38514F1FDF4B4F40D2181B3681C364BA0273C706
04017232BA853A7E731AF129F22FF4149563A419C26BF50A4C9D6EEFAD612601DB537DECE819B7F70F555A67C427A8CD9BF18AEB9B56E0C11056FAE6A3
0713612DCDDCB40AAB946BDA29CA91F73AF958AFD9
AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F0
043AE9E58C82F63C30282E1FE7BBF43FA72C446AF6F4618129097E2C5667C2223A902AB5CA449D0084B7E5B3DE7CCC01C9
10B7B4D696E676875615175137C8A16FD0DA2211

POSSIBLE SECRETS
139454871199115825601409655107690713107041707059928031797758001454375765357722984094124368522288239833039114681648076688236921220737322672160740747771700911134550432053804647694904686120113087816240740184800477047157336662926249423571248823968542221753660143391485680840520336859458494803187341288580489525163
0401A57A6A7B26CA5EF52FCDB816479700B3ADC94ED1FE674C06E695BABA1D
2E2F85F5DD74CE983A5C4237229DAF8A3F35823BE
68363196144955700784444165611827252895102170888761442055095051287550314083023
000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
MQVwithSHA512KDFAndSharedInfo
7fffffffffffffffffffffff800000cfa7e8594377d414c03821bc582063
FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A63A3620FFFFFFFFFFFFFFFF
30470ad5a005fb14ce2d9dcd87e38bc7d1b1c5facbaecbe95f190aa7a31d23c4dbbcbe06174544401a5b2c020965d8c2bd2171d3668445771f74ba084d2029d83c1c158547f3a9f1a2715be23d51ae4d3e5a1f6a7064f316933a346d3f529252
90EAF4D1AF0708B1B612FF35E0A2997EB9E9D263C9CE659528945C0D
BD71344799D5C7FCDC45B59FA3B9AB8F6A948BC5
c49d360886e704936a6678e1139d26b7819f7e90
00F50B028E4D696E676875615175290472783FB1
9B9F605F5A858107AB1EC85E6B41C8AA582CA3511EDDFB74F02F3A6598980BB9

POSSIBLE SECRETS
036768ae8e18bb92cfcf005c949aa2c6d94853d0e660bbf854b1c9505fe95a
617fab6832576cbbfed50d99f0249c3fee58b94ba0038c7ae84c8c832f2c
5EEEFCA380D02919DC2C6558BB6D8A5D
71FE1AF926CF847989EFEF8DB459F66394D90F32AD3F15E8
3086d221a7d46bcde86c90e49284eb15
324A6EDDD512F08C49A99AE0D3F961197A76413E7BE81A400CA681E09639B5FE12E59A109F78BF4A373541B3B9A1
044BA30AB5E892B4E1649DD0928643ADCD46F5882E3747DEF36E956E97
5FF6108462A2DC8210AB403925E638A19C1455D21
B4050A850C04B3ABF54132565044B0B7D7BFD8BA270B39432355FFB4
95475cf5d93e596c3fcd1d902add02f427f5f3c7210313bb45fb4d5bb2e5fe1cbd678cd4bbdd84c9836be1f31c0777725aeb6c2fc38b85f48076fa76bcd8146cc89a6fb2f706dd719898c2083dc8d896f84062e2c9c94d137b054a8d8096adb8d51952398eeca852a0af12df83e475aa65d4ec0c38a9560d5661186ff98b9fc9eb60eee8b030376b236bc73be3acdbd74fd61c1d2475fa3077b8f080467881ff7e1ca56fee066d79506ade51edbb5443a563927dbc4ba520086746175c8885925ebc64c6147906773496990cb714ec667304e261faee33b3cbdf008e0c3fa90650d97d3909c9275bf4ac86ffcb3d03e6dfc8ada5934242dd6d3bcca2a406cb0b
FFFFFFFFFFFFFFFFADF85458A2BB4A9AAFDC5620273D3CF1D8B9C583CE2D3695A9E13641146433FBCC939DCE249B3EF97D2FE363630C75D8F681B202AEC4617AD3DF1ED5D5FD65612433F51F5F066ED0856365553DED1AF3B557135E7F57C935984F0C70E0E68B77E2A689DAF3EFE8721DF158A136ADE73530ACCA4F483A797ABC0AB182B324FB61D108A94BB2C8E3FBB96ADAB760D7F4681D4F42A3DE394DF4AE56EDE76372BB190B07A7C8EE0A6D709E02FCE1CDF7E2ECC03404CD28342F619172FE9CE98583FF8E4F1232EEF28183C3FE3B1B4C6FAD733BB5FCBC2EC22005C58EF1837D1683B2C6F34A26C1B2EFFA886B423861285C97FFFFFFFFFFFFFFFF
6EE3CEEB230811759F20518A0930F1A4315A827DAC
c469684435deb378c4b65ca9591e2a5763059a2e

POSSIBLE SECRETS
9cdbd84c9f1ac2f38d0f80f42ab952e7338bf511
9DEF3CAFB939277AB1F12A8617A47BBBDBA51DF499AC4C80BEEEA9614B19CC4D5F4F5F556E27CBDE51C6A94BE4607A291558903BA0D0F84380B655BB9A22E8DCD F028A7CEC67F0D08134B1C8B97989149B609E0BE3BAB63D47548381DBC5B1FC764E3F4B53DD9DA1158BFD3E2B9C8CF56EDF019539349627DB2FD53D24B7C48665 772E437D6C7F8CE442734AF7CCB7AE837C264AE3A9BEB87F8A2FE9B8B5292E5A021FFF5E91479E8CE7A28C2442C6F315180F93499A234DCF76E3FED135F9BB
962eddcc369cba8ebb260ee6b6a126d9346e38c5
9162fbe73984472a0a9d0590
1053CDE42C14D696E67687561517533BF3F83345
03CE10490F6A708FC26DFE8C3D27C4F94E690134D5BFF988D8D28AAEAEDE975936C66BAC536B18AE2DC312CA493117DAA469C640CAF3
AADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA70330870553E5C414CA92619418661197FAC10471DB1D381085DDADDB58796829CA90069
027d29778100c65a1da1783716588dce2b8b4aee8e228f1896
047B6AA5D85E572983E6FB32A7CDEBC14027B6916A894D3AEE7106FE805FC34B44
1270212482889324174659070427771764435257876535089165358128175072657050312609850984974231883334834011809259999951209889341306592056149967242541210492743493570749203127695614516892241105793112488126102296785346384016935200132889950003622606842227508135323070045173416336850045410625869714168836867788425378203833
023809B2B7CC1B28CC5A87926AAD83FD28789E81E2C9E3BF10
b8adf1378a6eb73409fa6c9c637ba7f5
3d84f26c12238d7b4f3d516613c1759033b1a5800175d0b1
04A8C7DD22CE28268B39B55416F0447C2FB77DE107DCD2A62E880EA53EEB62D57CB4390295DBC9943AB78696FA504C11

POSSIBLE SECRETS
040060F05F658F49C1AD3AB1890F7184210EFD0987E307C84C27ACCFB8F9F67CC2C460189EB5AAAA62EE222EB1B35540CFE902374601E369050B7C4E42ACBA1DACB F04299C3460782F918EA427E6325165E9EA10E3DA5F6C42E9C55215AA9CA27A5863EC48D8E0286B
DC9203E514A721875485A529D2C722FB187BC8980EB866644DE41C68E143064546E861C0E2C9EDD92ADE71F46FCF50FF2AD97F951FDA9F2A2EB6546F39689BD3
038D16C2866798B600F9F08BB4A8E860F3298CE04A5798
04C0A0647EAAB6A48753B033C56CB0F0900A2F5C4853375FD614B690866ABD5BB88B5F4828C1490002E6773FA2FA299B8F
5037EA654196CFF0CD82B2C14A2FCF2E3FF8775285B545722F03EACDB74B
c39c6c3b3a36d7701b9c71a1f5804ae5d0003f4
1AB597A5B4477F59E39539007C7F977D1A567B92B043A49C6B61984C3FE3481AAF454CD41BA1F051626442B3C10
2866537B676752636A68F56554E12640276B649EF7526267
E8C2505DEDFC86DDC1BD0B2B6667F1DA34B82574761CB0E879BD081CFD0B6265EE3CB090F30D27614CB4574010DA90DD862EF9D4EBEE4761503190785A71C760
9CA8B57A934C54DEEDA9E54A7BBAD95E3B2E91C54D32BE0B9DF96D8D35
04009D73616F35F4AB1407D73562C10F00A52830277958EE84D1315ED31886
FFFFFFFE0000000075A30D1B9038A115
033C258EF3047767E7EDE0F1FDAA79DAEE3841366A132E163ACED4ED2401DF9C6BDCDE98E8E707C07A2239B1B097
040356DCD8F2F95031AD652D23951BB366A80648F06D867940A5366D9E265DE9EB240F
fca682ce8e12caba26efccf7110e526db078b05edecbcd1eb4a208f3ae1617ae01f35b91a47e6df63413c5e12ed0899bcd132acd50d99151bdc43ee737592e17

POSSIBLE SECRETS
00E8BEE4D3E2260744188BE0E9C723
A59A749A11242C58C894E9E5A91804E8FA0AC64B56288F8D47D51B1EDC4D65444FECA0111D78F35FC9FDD4CB1F1B79A3BA9CBEE83A3F811012503C8117F98E5048 B089E387AF6949BF8784EBD9EF45876F2E6A5A495BE64B6E770409494B7FEE1DBB1E4B2BC2A53D4F893D418B7159592E4FFFDF6969E91D770DAEBD0B5CB14C00AD 68EC7DC1E5745EA55C706C4A1C5C88964E34D09DEB753AD418C1AD0F4FDFD049A955E5D78491C0B7A2F1575A008CCD727AB376DB6E695515B05BD412F5B8C2F4C 77EE10DA48ABD53F5DD498927EE7B692BBBCDA2FB23A516C5B4533D73980B2A3B60E384ED200AE21B40D273651AD6060C13D97FD69AA13C5611A51B9085
C49D360886E704936A6678E1139D26B7819F7E90
3826F008A8C51D7B95284D9D03FF0E00CE2CD723A
04B8266A46C55657AC734CE38F018F2192
03F7061798EB99E238FD6F1BF95B48FEEB4854252B
D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF
b869c82b35d70e1b1ff91b28e37a62ecdc34409b
010090512DA9AF72B08349D98A5DD4C7B0532ECA51CE03E2D10F3B7AC579BD87E909AE40A6F131E9CFCE5BD967
115792089210356248762697446949407573529996955224135760342422259061068512044369
D7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F
F1FD178C0B3AD58F10126DE8CE42435B53DC67E140D2BF941FFDD459C6D655E1
0370F6E9D04D289C4E89913CE3530BFDE903977D42B146D539BF1BDE4E9C92
e8b4011604095303ca3b8099982be09fcb9ae616

POSSIBLE SECRETS
8e722de3125bddb05580164bfe20b8b432216a62926c57502ceede31c47816edd1e89769124179d0b695106428815065
02F40E7E2221F295DE297117B7F3D62F5C6A97FFCB8CEFF1CD6BA8CE4A9A18AD84FFABBD8EFA59332BE7AD6756A66E294AFD185A78FF12AA520E4DE739BACA0C7F FEFF7F2955727A
E4E6DB2995065C407D9D39B8D0967B96704BA8E9C90B
MQVwithSHA256KDFAndSharedInfo
0101BAF95C9723C57B6C21DA2EFF2D5ED588BDD5717E212F9D
C2173F1513981673AF4892C23035A27CE25E2013BF95AA33B22C656F277E7335
A9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5374
1854BEBDC31B21B7AEFC80AB0ECD10D5B1B3308E6DBF11C1
1C97BEFC54BD7A8B65ACF89F81D4D4ADC565FA45
0479BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B 8
10C0FB15760860DEF1EEF4D696E676875615175D
6C01074756099122221056911C77D77E77A777E7E7E77FCB
04B199B13B9B34EFC1397E64BAEB05ACC265FF2378ADD6718B7C7C1961F0991B842443772152C9E0AD
14201174159756348119636828602231808974327613839524373876287257344192745939351271897363116607846760036084894662356762579528277471921224 19290710461342083806363940845126918288940005715246254452957693493567527289568315417754417631393844571917550968471078465956625479423122 93338483924514339614727760681880609734239

POSSIBLE SECRETS
4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5
036b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296
662C61C430D84EA4FE66A7733D0B76B7BF93EBC4AF2F49256AE58101FEE92B04
0257927098FA932E7C0A96D3FD5B706EF7E5F5C156E16B7E7C86038552E91D
0400D9B67D192E0367C803F39E1A7E82CA14A651350AAE617E8F01CE94335607C304AC29E7DEFBD9CA01F596F927224CDECF6C
74D59FF07F6B413D0EA14B344B20A2DB049B50C3
686479766013060971498190079908139321726943530014330540939446345918554318339765605212255964066145455497729631139148085803712198799971664381257402829111505715 1
FFFFFFFFFFFFFFFFADF85458A2BB4A9AAFDC5620273D3CF1D8B9C583CE2D3695A9E13641146433FBCC939DCE249B3EF97D2FE363630C75D8F681B202AEC4617AD3DF1ED5D5FD65612433F51F5F066ED0856365553DED1AF3B557135E7F57C935984F0C70E0E68B77E2A689DAF3EFE8721DF158A136ADE73530ACCA4F483A797ABC0AB182B324FB61D108A94BB2C8E3FBB96ADAB760D7F4681D4F42A3DE394DF4AE56EDE76372BB190B07A7C8EE0A6D709E02FCE1CDF7E2ECC03404CD28342F619172FE9CE98583FF8E4F1232EEF28183C3FE3B1B4C6FAD733BB5FCBC2EC22005C58EF1837D1683B2C6F34A26C1B2EFFA886B4238611FCFDCDE355B3B6519035BBC34F4DEF99C023861B46FC9D6E6C9077AD91D2691F7F7EE598CB0FAC186D91CAEFE130985139270B4130C93BC437944F4FD4452E2D74DD364F2E21E71F54BFF5CAE82AB9C9DF69EE86D2BC522363A0DABC521979B0DEADA1DBF9A42D5C4484E0ABCD06BFA53DDEF3C1B20EE3FD59D7C25E41D2B669E1EF16E6F52C3164DF4FB7930E9E4E58857B6AC7D5F42D69F6D187763CF1D5503400487F55BA57E31CC7A7135C886EFB4318AED6A1E012D9E6832A907600A918130C46DC778F971AD0038092999A333CB8B7A1A1DB93D7140003C2A4ECEA9F98D0ACC0A8291CDCEC97DCF8EC9B55A7F88A46B4DB5A851F44182E1C68A007E5E655F6AFFFFFFFFFFFFFFFF
e4437ed6010e88286f547fa90abfe4c42212
F1FD178C0B3AD58F10126DE8CE42435B3961ADBCABC8CA6DE8FCF353D86E9C00
D7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FC

POSSIBLE SECRETS
AC6BDB41324A9A9BF166DE5E1389582FAF72B6651987EE07FC3192943DB56050A37329CBB4A099ED8193E0757767A13DD52312AB4B03310DCD7F48A9DA04FD50E8083969EDB767B0CF6095179A163AB3661A05FBD5FAAAE82918A9962F0B93B855F97993EC975EEAA80D740ADBF4FF747359D041D5C33EA71D281E446B14773BCA97B43A23FB801676BD207A436C6481F1D2B9078717461A5B9D32E688F87748544523B524B0D57D5EA77A2775D2ECFA032CFBDBF52FB3786160279004E57AE6AF874E7303CE53299CCC041C7BC308D82A5698F3A8D0C38271AE35F8E9DBFBB694B5C803D89F7AE435DE236D525F54759B65E372FCD68EF20FA7111F9E4AFF73
9760508f15230bccb292b982a2eb840bf0581cf5
3FA8124359F96680B83D1C3EB2C070E5C545C9858D03ECFB744BF8D717717EFC
13D56FFAEC78681E68F9DEB43B35BEC2FB68542E27897B79
7D5A0975FC2C3057EEF67530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9
51DEF1815DB5ED74FCC34C85D709
517cc1b727220a94fe13abe8fa9a6ee0

SCAN LOGS

Timestamp	Event	Error
2026-01-26 15:02:41	Generating Hashes	OK
2026-01-26 15:02:41	Extracting APK	OK

2026-01-26 15:02:41	Unzipping	OK
2026-01-26 15:02:41	Parsing APK with androguard	OK
2026-01-26 15:02:41	Extracting APK features using aapt/aapt2	OK
2026-01-26 15:02:42	Getting Hardcoded Certificates/Keystores	OK
2026-01-26 15:02:45	Parsing AndroidManifest.xml	OK
2026-01-26 15:02:45	Extracting Manifest Data	OK
2026-01-26 15:02:45	Manifest Analysis Started	OK
2026-01-26 15:02:45	Performing Static Analysis on: masvs_crypto (com.dkronig.masvs_crypto)	OK
2026-01-26 15:02:45	Fetching Details from Play Store: com.dkronig.masvs_crypto	OK
2026-01-26 15:02:46	Checking for Malware Permissions	OK
2026-01-26 15:02:46	Fetching icon path	OK

2026-01-26 15:02:46	Library Binary Analysis Started	OK
2026-01-26 15:02:46	Reading Code Signing Certificate	OK
2026-01-26 15:02:46	Running APKiD 3.0.0	OK
2026-01-26 15:02:48	Detecting Trackers	OK
2026-01-26 15:02:51	Decompiling APK to Java with JADX	OK
2026-01-26 15:03:42	Converting DEX to Smali	OK
2026-01-26 15:03:42	Code Analysis Started on - java_source	OK
2026-01-26 15:03:43	Android SBOM Analysis Completed	OK
2026-01-26 15:03:46	Android SAST Completed	OK
2026-01-26 15:03:46	Android API Analysis Started	OK

2026-01-26 15:04:19	Android API Analysis Completed	OK
2026-01-26 15:04:19	Android Permission Mapping Started	OK
2026-01-26 15:04:20	Android Permission Mapping Completed	OK
2026-01-26 15:04:21	Android Behaviour Analysis Started	OK
2026-01-26 15:04:23	Android Behaviour Analysis Completed	OK
2026-01-26 15:04:23	Extracting Emails and URLs from Source Code	OK
2026-01-26 15:04:23	Email and URL Extraction Completed	OK
2026-01-26 15:04:23	Extracting String data from APK	OK
2026-01-26 15:04:24	Extracting String data from Code	OK
2026-01-26 15:04:24	Extracting String values and entropies from Code	OK

2026-01-26 15:04:28	Performing Malware check on extracted domains	OK
2026-01-26 15:04:28	Saving to Database	OK

Report Generated by - MobSF v4.4.5

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

MobSF

ANDROID STATIC ANALYSIS REPORT

app_icon

masvs_platform (1.0)

File Name:	masvs_platform_release.apk
Package Name:	com.dkronig.masvs_platform
Scan Date:	Jan. 26, 2026, 4:13 p.m.
App Security Score:	52/100 (MEDIUM RISK)
Grade:	B

FINDINGS SEVERITY

HIGH	MEDIUM	INFO	SECURE	HOTSPOT
1	5	0	1	0

FILE INFORMATION

File Name: masvs_platform_release.apk
Size: 12.71MB
MD5: 5ee5a97e39dd75eeafe45dcb38cf92ca
SHA1: 2540252e76c4dc737a481478824f4e1fe6982929
SHA256: 43b15859919bc5e4fb795269fd78e4fca31d5033f30aaa943102ba1d70b628bd

APP INFORMATION

App Name: masvs_platform
Package Name: com.dkronig.masvs_platform
Main Activity: com.dkronig.masvs_platform.PlatformMenu
Target SDK: 35
Min SDK: 35
Max SDK:
Android Version Name: 1.0
Android Version Code: 1

APP COMPONENTS

Activities: 18
Services: 0
Receivers: 1
Providers: 2
Exported Activities: 0
Exported Services: 0
Exported Receivers: 1
Exported Providers: 1

CERTIFICATE INFORMATION

Binary is signed
v1 signature: False
v2 signature: True
v3 signature: False
v4 signature: False
X.509 Subject: CN=Dominic Kronig, OU=University, O=University of Bern, L=Bern, ST=Bern, C=3012
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2025-12-31 12:08:44+00:00
Valid To: 2050-12-25 12:08:44+00:00
Issuer: CN=Dominic Kronig, OU=University, O=University of Bern, L=Bern, ST=Bern, C=3012
Serial Number: 0x1
Hash Algorithm: sha256
md5: 71d59136f94aa3f5454bae353851cb7b
sha1: 62a94e3eabf027ca5dfdd42d912c2c5be84bd50b
sha256: 4f2afb4af5b8974656740ed44235ef4d3e4cff3b027afdde279058ad2fe13754
sha512: e7ed3abdb5695aed963f9c948b0be09ef3bbf09a1a410cd4b6efcd3e2b1d24e834d73a9642b15db205c81d71e539ea7bb933336d7248426d562089edb8bda536
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: 77428f4f4c526be66d196301e1335f4b576b37d16f9892b220b0ef75c37c346a
Found 1 unique certificates

APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO	DESCRIPTION
com.dkronig.masvs_platform.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION	unknown	Unknown permission	Unknown permission from android reference

APKID ANALYSIS

<table>
<tr><th>FILE</th><th>DETAILS</th></tr>
<tr><td>classes.dex</td><td>
<table>
<tr><th>FINDINGS</th><th>DETAILS</th></tr>
<tr><td>yara_issue</td><td>yara issue - dex file recognized by apkid but not yara module</td></tr>
<tr><td>Anti-VM Code</td><td>Build.FINGERPRINT check
Build.MANUFACTURER check</td></tr>
<tr><td>Compiler</td><td>unknown (please file detection issue!)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>FILE</th><th>DETAILS</th></tr>
<tr><td>classes2.dex</td><td>
<table>
<tr><th>FINDINGS</th><th>DETAILS</th></tr>
<tr><td>yara_issue</td><td>yara issue - dex file recognized by apkid but not yara module</td></tr>
<tr><td>Compiler</td><td>unknown (please file detection issue!)</td></tr>
</table>
</td></tr>
</table>

NETWORK SECURITY

NO	SCOPE	SEVERITY	DESCRIPTION

CERTIFICATE ANALYSIS

HIGH: **0** | WARNING: **0** | INFO: **1**

TITLE	SEVERITY	DESCRIPTION
Signed Application	info	Application is signed with a code signing certificate

MANIFEST ANALYSIS

HIGH: **1** | WARNING: **3** | INFO: **0** | SUPPRESSED: **0**

NO	ISSUE	SEVERITY	DESCRIPTION
1	Debug Enabled For App [android:debuggable=true]	high	Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.
2	Application Data can be Backed up [android:allowBackup] flag is missing.	warning	The flag [android:allowBackup] should be set to false. By default it is set to true and allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
3	Content Provider (com.dkronig.masvs_platform.maswe_0064.CustomContentProvider) is not Protected. [android:exported=true]	warning	A Content Provider is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.
4	Broadcast Receiver (androidx.profileinstaller.ProfileInstallReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.DUMP [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

CODE ANALYSIS

HIGH: **0** | WARNING: **1** | INFO: **0** | SECURE: **0** | SUPPRESSED: **0**

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	CWE: CWE-312: Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14	com/dkronig/common/BaseLoginActivity.java com/dkronig/common/BaseRegisterActivity.java

NIAP ANALYSIS v1.3

NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION

BEHAVIOUR ANALYSIS

RULE ID	BEHAVIOUR	LABEL	FILES
00125	Check if the given file path exist	file	com/dkronig/common/BaseLoginActivity.java

ABUSED PERMISSIONS

TYPE	MATCHES	PERMISSIONS
Malware Permissions	0/25	
Other Common Permissions	0/44	

Malware Permissions:
Top permissions that are widely abused by known malware.
Other Common Permissions:
Permissions that are commonly abused by known malware.

HARDCODED SECRETS

POSSIBLE SECRETS
39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319
6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151
39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308152294913554433653942643
051953eb9618e1c9a1f929a21a0b68540eea2da725b99b315f3b8b489918ef109e156193951ec7e937b1652c0bd3bb1bf073573df883d2c34f1ef451fd46b503f00
115792089210356248762697446949407573530086143415290314195533631308867097853951
6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296
11839296a789a3bc0045c8a5fb42c7d1bd998f54449579b446817afbd17273e662c97ee72995ef42640c550b9013fad0761353c7086a272c24088be94769fd16650
3617de4a96262c6f5d9e98bf9292dc29f8f41dbd289a147ce9da3113b5f0b8c00a60b1ce1d7e819d7a431d7c90ea0e5f
4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5
c6858e06b70404e9cd9e3ecb662395b4429c648139053fb521f828af606b4d3dbaa14b5e77efe75928fe1dc127a2ffa8de3348b3c1856a429bf97e7e31c2e5bd66
b3312fa7e23ee7e4988e056be3f82d19181d9c6efe8141120314088f5013875ac656398d8a2ed19d2a85c8edd3ec2aef

POSSIBLE SECRETS
aa87ca22be8b05378eb1c71ef320ad746e1d3b628ba79b9859f741e082542a385502f25dbf55296c3a545e3872760ab7
115792089210356248762697446949407573529996955224135760342422259061068512044369
5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
686479766013060971498190079908139321726943530014330540939446345918554318339765539424505774633321719753296399637136332111386476861244038034037280889270700 5449

SCAN LOGS

Timestamp	Event	Error
2026-01-26 17:32:16	Generating Hashes	OK
2026-01-26 17:32:16	Extracting APK	OK
2026-01-26 17:32:16	Unzipping	OK
2026-01-26 17:32:16	Parsing APK with androguard	OK
2026-01-26 17:32:17	Extracting APK features using aapt/aapt2	OK

2026-01-26 17:32:17	Getting Hardcoded Certificates/Keystores	OK
2026-01-26 17:32:17	Parsing AndroidManifest.xml	OK
2026-01-26 17:32:17	Extracting Manifest Data	OK
2026-01-26 17:32:17	Manifest Analysis Started	OK
2026-01-26 17:32:17	Performing Static Analysis on: masvs_platform (com.dkronig.masvs_platform)	OK
2026-01-26 17:32:18	Fetching Details from Play Store: com.dkronig.masvs_platform	OK
2026-01-26 17:32:18	Checking for Malware Permissions	OK
2026-01-26 17:32:18	Fetching icon path	OK
2026-01-26 17:32:18	Library Binary Analysis Started	OK
2026-01-26 17:32:18	Reading Code Signing Certificate	OK
2026-01-26 17:32:19	Running APKiD 3.0.0	OK

2026-01-26 17:32:20	Detecting Trackers	OK
2026-01-26 17:32:23	Decompiling APK to Java with JADX	OK
2026-01-26 17:33:02	Converting DEX to Smali	OK
2026-01-26 17:33:02	Code Analysis Started on - java_source	OK
2026-01-26 17:33:02	Android SBOM Analysis Completed	OK
2026-01-26 17:33:36	Android SAST Completed	OK
2026-01-26 17:33:36	Android API Analysis Started	OK
2026-01-26 17:33:38	Android API Analysis Completed	OK
2026-01-26 17:33:38	Android Permission Mapping Started	OK
2026-01-26 17:33:39	Android Permission Mapping Completed	OK
2026-01-26 17:33:39	Android Behaviour Analysis Started	OK

2026-01-26 17:33:41	Android Behaviour Analysis Completed	OK
2026-01-26 17:33:41	Extracting Emails and URLs from Source Code	OK
2026-01-26 17:33:42	Email and URL Extraction Completed	OK
2026-01-26 17:33:42	Extracting String data from APK	OK
2026-01-26 17:33:42	Extracting String data from Code	OK
2026-01-26 17:33:42	Extracting String values and entropies from Code	OK
2026-01-26 17:33:45	Performing Malware check on extracted domains	OK
2026-01-26 17:33:45	Saving to Database	OK

Report Generated by - MobSF v4.4.5

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

B. Full Semgrep Scan Results

Semgrep CLI

Scanning 332 files with 86 Code rules:

CODE RULES

Language	Rules	Files
java	82	150
kotlin	3	7

Origin	Rules
Community	86

SUPPLY CHAIN RULES

💎 Sign in with `semgrep login` and run `semgrep ci` to find dependency vulnerabilities and advanced cross-file findings.

PROGRESS

100% 0:00:00

19 Code Findings

masvs_crypto\src\main\java\com\dkronig\masvs_crypto\maswe_0009\EncryptionHandler.java

❯❯ java.lang.security.audit.crypto.desede-is-deprecated.desede-is-deprecated

Triple DES (3DES or DESede) is considered deprecated. AES is the recommended cipher. Upgrade to use

AES.

Details: https://sg.run/Geqn

```
 89┆ KeyGenerator keyGenerator = KeyGenerator.getInstance(ALGORITHM_DES);
```

❯❯ java.lang.security.audit.crypto.des-is-deprecated.des-is-deprecated

DES is considered deprecated. AES is the recommended cipher. Upgrade to use AES. See

https://www.nist.gov/news-events/news/2005/06/nist-withdraws-outdated-data-encryption-standard for

more information.

Details: https://sg.run/5Q73

```
 ▶▶┆ Autofix ▶ "AES/GCM/NoPadding"
 89┆ KeyGenerator keyGenerator = KeyGenerator.getInstance(ALGORITHM_DES);
  ⋮┆----------------------------------------
 ▶▶┆ Autofix ▶ "AES/GCM/NoPadding"
118┆ Cipher cipher = Cipher.getInstance(CIPHER_TRANSFORMATION);
  ⋮┆----------------------------------------
 ▶▶┆ Autofix ▶ "AES/GCM/NoPadding"
133┆ Cipher cipher = Cipher.getInstance(CIPHER_TRANSFORMATION);
```

masvs_crypto\src\main\java\com\dkronig\masvs_crypto\maswe_0010\EncryptionHandler.java

❯❯ java.lang.security.audit.cbc-padding-oracle.cbc-padding-oracle

Using CBC with PKCS5Padding is susceptible to padding oracle attacks. A malicious actor could

discern the difference between plaintext with valid or invalid padding. Further, CBC mode does not

```
    include any integrity checks. Use 'AES/GCM/NoPadding' instead.
    Details: https://sg.run/ydxr

     ▶▶┆ Autofix ▶ "AES/GCM/NoPadding"
     64┆ Cipher cipher = Cipher.getInstance(CIPHER_TRANSFORMATION);
      ⋮┆----------------------------------------
     ▶▶┆ Autofix ▶ "AES/GCM/NoPadding"
    102┆ Cipher cipher = Cipher.getInstance(CIPHER_TRANSFORMATION);


masvs_crypto\src\main\java\com\dkronig\masvs_crypto\maswe_0021\EncryptionHandler.java
  ❯❯ java.lang.security.audit.crypto.use-of-sha1.use-of-sha1
    Detected SHA1 hash algorithm which is considered insecure. SHA1 is not collision resistant
and is
    therefore not suitable as a cryptographic signature. Instead, use PBKDF2 for password
hashing or
    SHA256 or SHA512 for other hash function applications.
    Details: https://sg.run/bXNp

     24┆ MessageDigest digestAlgorithm =
MessageDigest.getInstance(HASHING_ALGORITHM);


masvs_crypto\src\main\java\com\dkronig\masvs_crypto\maswe_0022\EncryptionHandler.java
  ❯❯ java.lang.security.audit.crypto.no-static-initialization-vector.no-static-initialization-vector
    Initialization Vectors (IVs) for block ciphers should be randomly generated each time they
are used.
    Using a static IV means the same plaintext encrypts to the same ciphertext every time,
weakening the
    strength of the encryption.
    Details: https://sg.run/BkB5

     17┆ public class EncryptionHandler {
     18┆     private static final String SECRET_KEY_ALIAS = "maswe_0022_secret_key";
```

```
    private static final String ENCRYPTION_ALGORITHM = "AES";
    private static final String CIPHER_TRANSFORMATION = "AES/CBC/PKCS5PADDING";
    private static final String ENCRYPTION_KEY = "encryption_key";
    private static final int KEY_SIZE = 256;

    private static SharedPreferences sharedPreferences;
    private static byte[] iv = {47, -98, 3, 120, 14, -55, 89, 6, -12, 33, 9, -44, 63, -1,
  77, 22};

  [hid 121 additional lines, adjust with --max-lines-per-finding]
```

masvs_platform\src\main\java\com\dkronig\masvs_platform\maswe_0053\EncryptionHandler.java

❱ java.lang.security.audit.crypto.gcm-detection.gcm-detection

GCM detected, please check that IV/nonce is not reused, an Initialization Vector (IV) is a nonce
used to randomize the encryption, so that even if multiple messages with identical plaintext are
encrypted, the generated corresponding ciphertexts are different. Unlike the Key, the IV usually
does not need to be secret, rather it is important that it is random and unique. Certain encryption
schemes the IV is exchanged in public as part of the ciphertext. Reusing same Initialization Vector
with the same Key to encrypt multiple plaintext blocks allows an attacker to compare the ciphertexts
and then, with some assumptions on the content of the messages, to gain important information about
the data being encrypted.

Details: https://sg.run/BLLb

```
  Cipher cipher = Cipher.getInstance(AES_MODE);
 ⋮┆----------------------------------------
```

```
88┆ Cipher cipher = Cipher.getInstance(AES_MODE);
⋮┆----------------------------------------
89┆ GCMParameterSpec spec = new GCMParameterSpec(GCM_TAG_LENGTH, iv);
```

masvs_platform\src\main\java\com\dkronig\masvs_platform\maswe_0055\EncryptionHandler.java

❱ java.lang.security.audit.crypto.gcm-detection.gcm-detection

GCM detected, please check that IV/nonce is not reused, an Initialization Vector (IV) is a nonce

used to randomize the encryption, so that even if multiple messages with identical plaintext are

encrypted, the generated corresponding ciphertexts are different. Unlike the Key, the IV usually

does not need to be secret, rather it is important that it is random and unique. Certain encryption

schemes the IV is exchanged in public as part of the ciphertext. Reusing same Initialization Vector

with the same Key to encrypt multiple plaintext blocks allows an attacker to compare the ciphertexts

and then, with some assumptions on the content of the messages, to gain important information about

the data being encrypted.

Details: https://sg.run/BLLb

```
55┆ Cipher cipher = Cipher.getInstance(AES_MODE);
⋮┆----------------------------------------
87┆ Cipher cipher = Cipher.getInstance(AES_MODE);
⋮┆----------------------------------------
88┆ GCMParameterSpec spec = new GCMParameterSpec(GCM_TAG_LENGTH, iv);
```

masvs_platform\src\main\java\com\dkronig\masvs_platform\maswe_0067\EncryptionHandler.java

❱ java.lang.security.audit.crypto.gcm-detection.gcm-detection

GCM detected, please check that IV/nonce is not reused, an Initialization Vector (IV) is a nonce

used to randomize the encryption, so that even if multiple messages with identical plaintext are

encrypted, the generated corresponding ciphertexts are different. Unlike the Key, the IV usually

does not need to be secret, rather it is important that it is random and unique. Certain encryption

schemes the IV is exchanged in public as part of the ciphertext. Reusing same Initialization Vector

with the same Key to encrypt multiple plaintext blocks allows an attacker to compare the ciphertexts

and then, with some assumptions on the content of the messages, to gain important information about

the data being encrypted.

Details: https://sg.run/BLLb

```
55┆ Cipher cipher = Cipher.getInstance(AES_MODE);
⋮┆----------------------------------------
87┆ Cipher cipher = Cipher.getInstance(AES_MODE);
⋮┆----------------------------------------
88┆ GCMParameterSpec spec = new GCMParameterSpec(GCM_TAG_LENGTH, iv);
```

masvs_storage\src\main\java\com\dkronig\masvs_storage\maswe_0006\EncryptionHandler.java

❯❱ java.lang.security.audit.crypto.des-is-deprecated.des-is-deprecated

DES is considered deprecated. AES is the recommended cipher. Upgrade to use AES. See

https://www.nist.gov/news-events/news/2005/06/nist-withdraws-outdated-data-encryption-standard for

more information.

Details: https://sg.run/5Q73

```
▶▶┆ Autofix ▶ "AES/GCM/NoPadding"
26┆ Cipher cipher = Cipher.getInstance(ALGORITHM);
```

```
⋮┆----------------------------------------
▶▶┆ Autofix ▶ "AES/GCM/NoPadding"
42┆ Cipher cipher = Cipher.getInstance(ALGORITHM);


┌──────────────┐
│ Scan Summary │
└──────────────┘
✅ Scan completed successfully.
• Findings: 19 (19 blocking)
• Rules run: 82
• Targets scanned: 157
• Parsed lines: ~100.0%
• Scan skipped:
  ◦ Files larger than  files 1.0 MB: 1
  ◦ Files matching .semgrepignore patterns: 27
• For a detailed list of skipped files and lines, run semgrep with the --verbose flag
Ran 82 rules on 157 files: 19 findings.
💎 Missed out on 29 pro rules since you aren't logged in!
⚡ Supercharge Semgrep OSS when you create a free account at https://sg.run/rules.
```