KINGDOM OF SAUDI ARABIA
Ministry of Education
Al-Imam Mohammad University
College of Computer & Information Sciences

المملكة العربية السعودية
وزارة التعليم
جامعة الامام محمد بن سعود الإسلامية
كلية علوم الحاسب والمعلومات

Quality Assessment for Alinma Bank

Workshop 1

Abdullah Jamaan Baskran 440012378
Fahad Al Thenayyan 440012726
Mohammed Alawashiz 438020893

Supervisor
Dr. Sultan S. Alqahtnai

Table of Contents

1-Introduction:

1.1 Purpose:

The aim of this workshop is to discover methods for determining the quality of delivered programs, identifying vulnerabilities and security gaps, and determining how to prevent, solve, and make programs as safe as possible.

1.2 Goals:

The goal is to identify vulnerabilities and security gaps which include in points.

1- Determine the app's weaknesses.
2- Find bugs in the source code.
3- Obfuscate in the source code.

Finally, we'll talk about how to avoid and fix them.

1.3 Tools Used

- Mobsf
- SonarQube
- VisualCodeGrepper

2- Static analysis:

By using a SonarQube, mobsf and Visual Code Grepper tool we perform a static analysis for source code, apk file that what we found.

2.1 Bugs:

Bugs is error in program that causes to unexpected behavior. The following found in code.

1- Ignore the return operation status code from method

Ignoring return value may yield to arise in security risks when the invoking method fails to take suitable action Consequently, programs must not ignore method return values.

2- Identical expressions used on both sides of a binary operator

Using the same value on both side of binary operators is almost always mistake which lead to dead code.

3-Check equality between two object using operator '='

It's mistake to compare between tow Object by using binary operator since doesn't compare the actual value it compares the memory location.

4-Variables begin self-assigned

There is no reason to re-assign a variable to itself, the re-assignment is a mistake, and some other value or variable was intended for the assignment instead.

2.2 Vulnerability:

a vulnerability is a weakness that can be exploited by cybercriminals to gain unauthorized access to a computer system, that what we found by using SonarQube, mobsf, Visual Code Grepper.

1-Using a fixed Initialization Vector (IV)

When encrypting data with the Cipher Block Chaining (CBC) mode an Initialization Vector (IV) is used to randomize the encryption. When using fixed size make it predicable.

2 -Allow to use a weak SSL/TLS protocol

Using or allowing of insecure TLS protocols like (TLS 1.0, TLS 1.1) could open the door to downgrade attacks: a malicious actor who is able to intercept the connection could modify the requested protocol version and downgrade it to a less secure version.

3 - Cryptographic Algorithm usage of 'MD5'

The cryptographic algorithm in use is MD5 which is a weak hash known to have hash collisions. MD5 is considered weak and insecure; an attacker can easily use an MD5 collision to forge valid digital certificates. The most well-known example of this type of attack is when attackers forged a Microsoft Windows code-signing certificate and used it to sign the Flame malware.

4-Contents Provider

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.

5- Insecure Random Number Generator

Instances of java. util. Random are not cryptographically secure. And the Package is flawed and produces predictable values for any given seed which are reproducible once the starting seed is identified

2.3 Potential Unsafe code

We found with Visual Code Grepper some potential unsafe code which list here

1-The code appears to have classes that contain public variables which may be accessed and modified by other classes without the use of getter/setter methods. It is considered unsafe to have public fields or methods in a class as any method, field, or class that is not private is a potential avenue of attack. It is safer to provide accessor methods to variables in order to limit their accessibility.

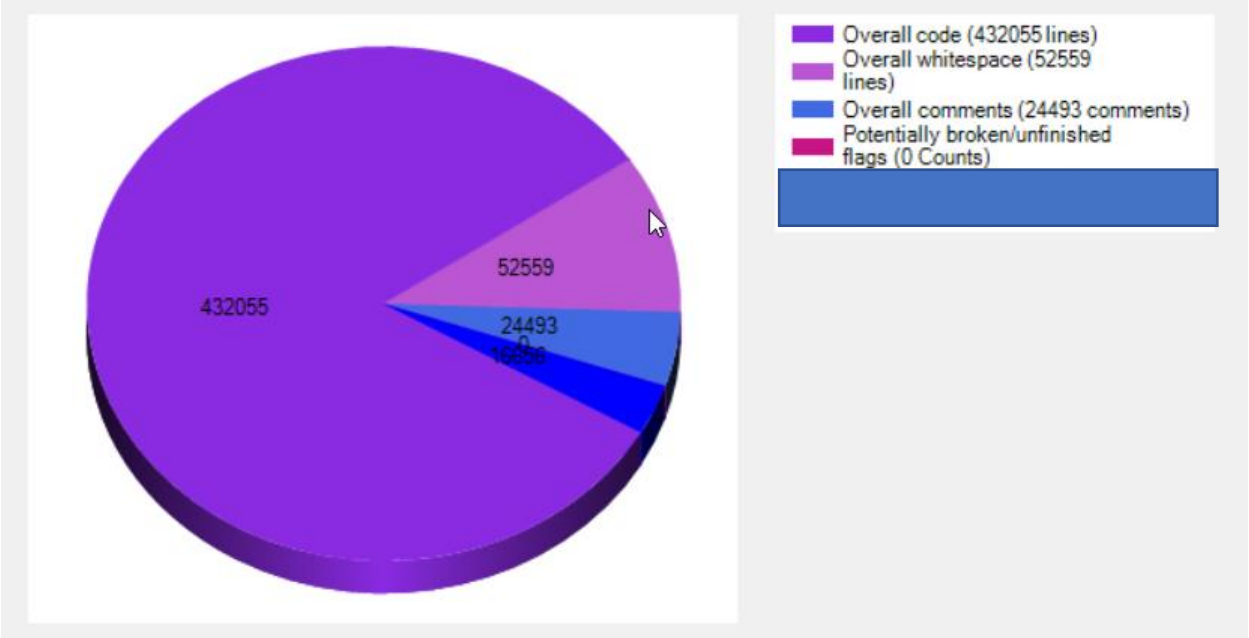
2- The code appears to have public classes that are not declared as final as per OWASP recommendation, it is consider best practice to make classes final. Non-Final classes can allow an attacker to extend a class in a malicious manner. its values can be maliciously manipulated by any function that has access to it in order to extend the application code or acquire critical information about the application.

3- The use of operations on Primitive data type. The code appears to be carrying out a mathematical operation on a primitive data type, in some circumstances this can result in an overflow or an unexpected behavior

2.4 Documentation

The system is not documented in the source code after pulling all the comments we found that none of them describes methods or classes with that in mind we can assume that the application is either not documented which is unlikely to be the case of it is externally documented and separated from the source code we analyzed

A breakdown of the code lines



There are 509704 lines of which there are:

~4.8% comments (24463)

~10.3% lines of white space (52559)

~84.9% lines of code(432652)

2.5 Obfuscate in the code:

Obfuscation is way to protect your code from reverse engineer by make it difficult to understand, so we made a program written in python to count the percentage of obfuscated file by name of the file, based on if file name contains on only one character so count it as obfuscated file or duplicate character, we except file R.java which it auto-generated file by AAPT (Android Asset Packaging Tool). See figure 1.

Number of files is 4481 files.

Number of obfuscated files is 1508 files.

Number of non-obfuscated files is 2973 files.

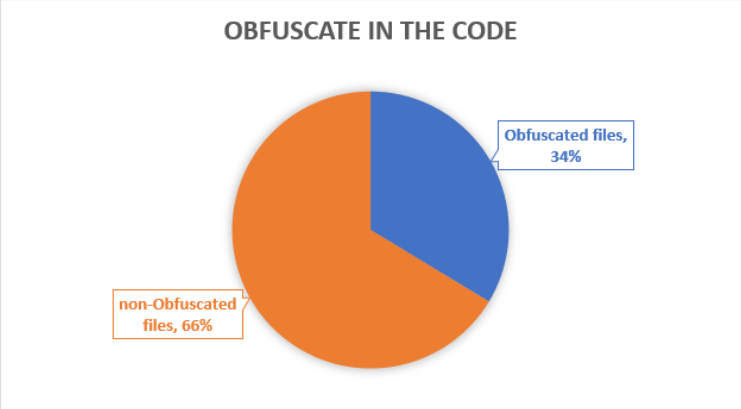


Figure 1. Obfusate in the code

3- Discussion:

3.1 Bugs:

Here will discuss the result of bug section and the solution of each problem.

1- Ignore the return operation status code from the method

Ignoring the status code led to a raise in error and security risks

which is undesirable, should avoid this situation, the best way to avoid it is to make sure you test any status code to see whether it is completed successfully or failed.

2- Identical expressions used on both sides of a binary operator

It's almost always a mistake to use the same value on both sides of a binary operator. It's either a copy/paste error and thus a bug with logical operators, or it's just wasted code that could be streamlined. Having the same value on both sides of a bitwise operator, as well as other binary mathematical operators, produces predictable results and should be simplified.

Solution is avoiding any usage of identical expressions that are not required or make sure what you want to make it

3- Check equality between two object using operator '='

Confusing reference equality and object equality can lead to unexpected results. Because the '==' and '!=' operators compare object references rather than object values, the values of object cannot be directly compared using these operators. The best way is using equals function to check the actual value not the memory location.

4-Variables begin self-assigned

When a field, a local, or a parameter symbol is assigned to itself. When a local, parameter, or field symbol has a name that is identical to another symbol in scope, this is a typical error. Rather than utilizing distinct symbols on the left and right sides of the assignment, the same symbol was used on both. This results in a duplicate assignment of the value to itself, which is usually a sign of a bug. Either the statement is redundant and should be removed if don't side effect.

3.2 Vulnerability:

1-Using a fixed Initialization Vector (IV)

Using Initialization Vector (IV) in a predictable way that happens when you are using fixed-size, makes it susceptible to a dictionary attack and Chosen-Plaintext Attack which is attacked to gain information that reduces the security of the encryption scheme. To avoid it you must use IV in an unpredictable way. There are two approaches for creating unexpected IVs that are approved. The first way is to encrypt a nonce using the forward cipher function using the same key that was used to encrypt the content. The nonce must be a data block that is different for each encryption operation execution. The nonce might be a counter or a message number, the second option is to use a secure random number generator to produce a random data block.

2 -Allow to use a weak SSL/TLS protocol:

Outdated TLS protocols utilize cipher suites that are no longer maintained or recommended and utilizing earlier TLS versions would necessitate extra effort to keep the libraries up to date, raising product maintenance costs.

TLS 1.0 and 1.1 are vulnerable to downgrade attacks since they rely on the SHA-1 hash to ensure message integrity. Even handshake authentication is done with SHA-1, making it easier for an attacker to impersonate a server in MITM attacks.

TLS 1.1 and earlier protocols lack the ability to use more robust hashing methods, which is available in subsequent protocols, to avoid such kind of problem is recommended to enforce TLS 1.2 as the minimum protocol version and to disallow older versions like TLS 1.0.

3- Cryptographic Algorithm usage of 'MD5'

The MD5 algorithm has weaknesses that allow for output collisions. As a result, attackers can produce cryptographic tokens or other data that look to be legitimate but aren't.

Don’t use 'MD5' instead OWASP recommends the use of one of these algorithms

- Confidentiality algorithms: AES-GCM-256 or ChaCha20-Poly1305
- Integrity algorithms: SHA-256, SHA-384, SHA-512, Blake2, the SHA-3 family
- Digital signature algorithms: RSA (3072 bits and higher), ECDSA with NIST P-384
- Key establishment algorithms: RSA (3072 bits and higher), DH (3072 bits or higher), ECDH with NIST P-384

4- Content Provider

We can either:

- change Set android:exported attribute’s value to false
- Limit access with custom permissions imposing permission-based restrictions by defining custom permissions for an activity.

5-Insecure Random Number Generator

When a function that can provide predictable values is utilized as a source of randomness in a security-sensitive situation, insecure randomness mistakes arise.

Computers are deterministic machines, which means they can't generate actual randomness. PRNGs (Pseudo-Random Number Generators) are algorithms that

approximate randomness by beginning with a seed and calculating subsequent values from it.

PRNGs are classified as statistical or cryptographic. Statistical PRNGs have useful statistical qualities, but their output is extremely predictable and creates an easy to replicate numeric stream, making them inappropriate for application in situations where produced numbers must be surprising.

Instead of using Random use Secure Random to get a cryptographically secure pseudo-random number generator for use by security-sensitive applications.

4- Conclusion

In the end, we achieved the purpose of this workshop using tools that make us discover the quality and security weaknesses of given app. And we have achieved our goals to find bugs and Vulnerabilities, we explain them in detail, and we discover ways to avoid them and alternative solutions, also we calculate the percentage of obfuscate in the code, we perform a breakdown of the code lines and some of potentially unsafe code.