Auditor: Maleak Teouri

2023 Security Assessment Report Prepared For

TRILLIONER

TLC Wallet (Android)

Date: 30/10/2023 - 03/11/2023

Report Informations

Project	
Mission	Black box penetration tests & audit
Client	Trillioner Innovation
Start Date	30/10/2023
End Date	03/11/2023

Disclaimer:

Note that this assessment may not disclose all vulnerabilities that are present on the application. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible on TLC Wallet's application. Any changes made to the application during the period of testing may affect the results of the assessment.

Contents

Summary

This document is a summary of the work I conducted. The subject of the test was the mobile application TLC Wallet for Android system available at:

- *https://play.google.com/store/apps/details?id=com.qfx.trillionerwallet&hl=en&gl=US&pli=1*

Tests were conducted without any sources or relevant information disclosed on the system **(black box assessment)**.

The most severe vulnerability identified during the assessment was:

- *Brute-force login mechanism*

During the tests, particular emphasis was placed on vulnerabilities that might lead to a loss of funds.

The security tests were carried out in accordance with generally accepted methodologies, including: OWASP TOP10, as well as commonly known good practices of security.

An approach based on manual tests (using the above-mentioned methodologies), supported by a number of analysis tools (i.a. Burp Suite, MobSF, JADX), was used during the assessment.

The vulnerabilities are described in detail in further parts of the report.

Severity classification

Vulnerabilities are classified in a five-point scale, that is reflecting both the probability of exploitation of the vulnerability and the business risk of its exploitation. The severity goes from High to Informational.

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	🟥 Critical	🟧 High	🟨 Medium
Likelihood: Medium	🟧 High	🟨 Medium	🟩 Low
Likelihood: Low	🟨 Medium	🟩 Low	🟩 Low

** Issues marked as '*🟦 Informational*' are not security vulnerabilities per se. Their aim is to point out good practices, the implementation of which will lead to the overall increase of the system security level.*

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Remediation Effort Classification

Difficulty	Description
🟧 Hard	Remediation may require extensive reconfiguration of underlying systems that is time consuming. Remediation may require disruption of normal business functions.
🟨 Moderate	Remediation may require minor reconfigurations or additions that may be time-intensive or expensive.
🟩 Easy	Remediation can be accomplished in a short amount of time, with little difficulty.

Findings overview

Total vulnerabilities	8
🟥 Critical	0
🟧 High	0
🟨 Medium	3
🟩 Low	0
🟦 Informational	5

Findings table

ID	Vulnerability title	Severity	Remediation effort
[M-01]	Brute-force login mechanism	🟧 Medium	🟩 Easy
[M-02]	Lack of Access Control to reveal seed phrases	🟧 Medium	🟩 Easy
[M-03]	Swap function not working due to deprecated API	🟧 Medium	🟩 Easy
[NC-01]	Weak password policy	🟦 Informational	🟩 Easy
[NC-02]	Misleading swap functionality	🟦 Informational	🟩 Easy
[NC-03]	Potential leak of private keys	🟦 Informational	🟩 Easy
[NC-04]	Missing rooted device detection	🟦 Informational	🟧 Moderate
[NC-05]	Certificate transparency not implemented	🟦 Informational	🟧 Moderate

Vulnerabilities

[M-01] Brute-force login mechanism risk

Severity	Remediation effort
🟧 Medium	🟩 Easy

Description

The login mechanism presents a vulnerability that makes it susceptible to brute-force attacks. It permits unlimited login attempts as password validation is performed on user input, without any validation constraints. In combination with a weak password policy (see [NC-01] Weak password policy), it further elevates the risk of unauthorized access to the user's wallet.

Risk

Likelihood: Low - An attacker has to gain physical or remote access (i.e. via RAT) to the victim's phone.

Impact: High - An unauthorized access may lead to the theft of all user's funds.

Evidence

The application logs if the password is incorrect on keyboard input and not on login submit (see figure below).

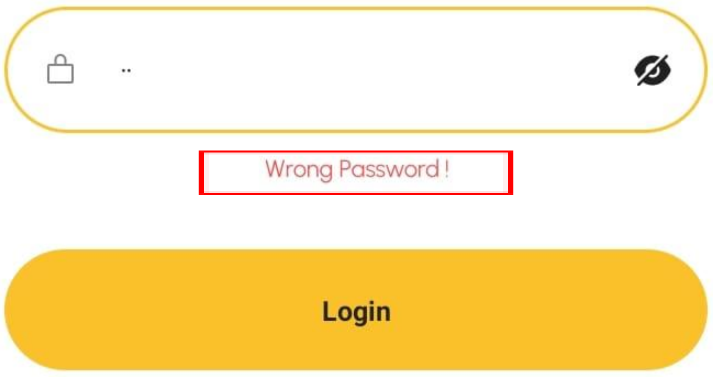


Figure 1 - Login mechanism

**"Wrong Password!" was displayed without submitting the password*

In combination with a weak password policy (which allows “12345678” as a password) it becomes very likely for an attacker to crack the password. With an emulated keyboard and a brute force algorithm, the hack can be performed in a few seconds.

Remediation

It is recommended to check the password on submit rather than on input. Additionally, implement an account lockout policy that will temporarily lock the user's account after a certain number of failed login attempts.

It is also recommended to implement a strong password policy (see [NC-01] Weak password policy)

Reference:
https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

[M-02] Lack of Access Control to reveal seed phrases

Severity	Remediation effort
🟧 Medium	🟩 Easy

Description

For a crypto wallet, seed phrases are equivalent to a private key. Stealing them means acquiring control over crypto funds. Therefore, extreme precautions must be taken while dealing with these seed phrases, especially when revealing them. Here, the application does not perform any authorization check before revealing them on a logged user. Although it requires a user to already have logged into the application, a plausible scenario where an unauthorized user has access to the logged account (i.e. a close relative) introduces a significant security concern.

Risk

Likelihood: Low - Attacker needs access to a logged account.

Impact: High - An unauthorized access to the seed phrases may lead to the loss of all user's funds.

Evidence

On this activity, no checks were required before displaying the seed phrases:

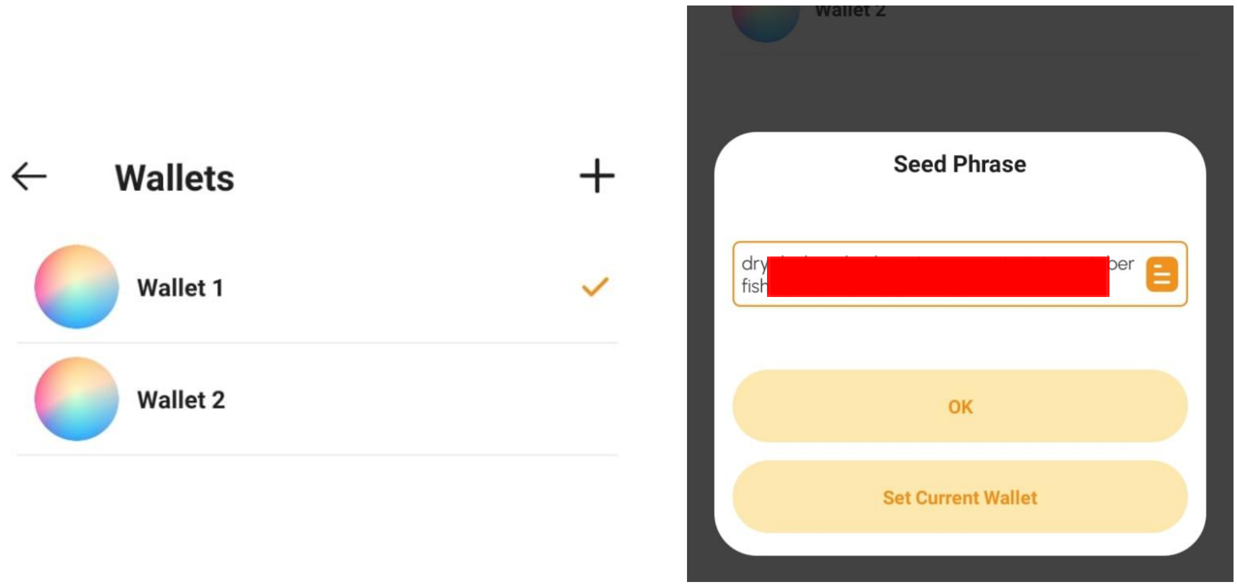


Figure 2 - No access control to reveal seed phrases

Remediation

To mitigate this security risk, it is imperative to implement access control measures that include a password check before revealing seed phrases. This additional layer of protection will enhance the security of seed phrase access.

Furthermore, it is crucial to raise user awareness regarding the criticality of seed phrases. Educate users about the importance of safeguarding this information and the potential risks associated with its exposure. This proactive approach will contribute to a more secure and informed user base.

[M-03] Swap function not working due to deprecated API

Severity	Remediation effort
🟧 Medium	🟩 Easy

Description

Quote for a swap returns a 1:0 ratio for every token. As a result, the swap function is not working as it requires an amount to swap greater than zero. Further analysis revealed that the quote fetching process relies on the deprecated api.1inch.io.

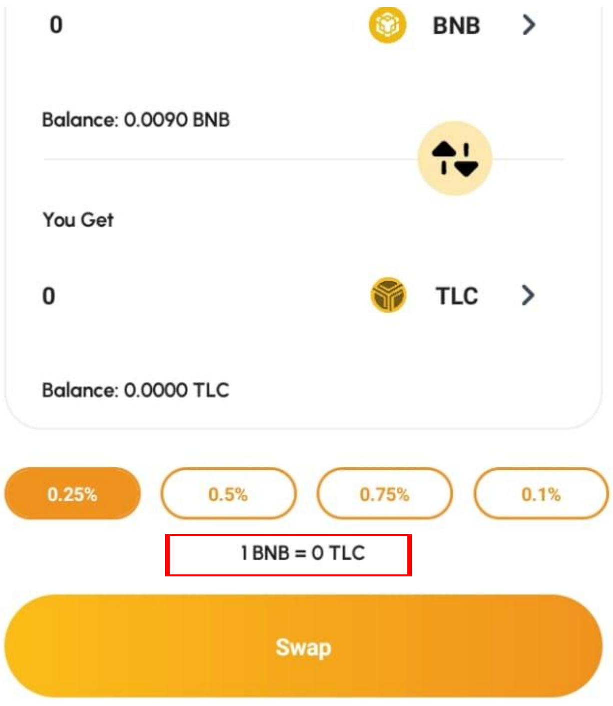
Risk

Likelihood: High - Not fixed until application update.

Impact: Low - Loss of functionality.

Technical details

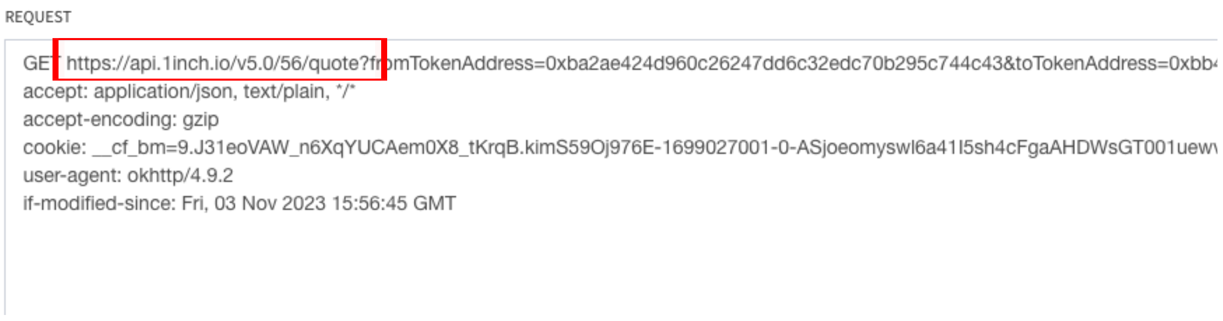
The quote to swap BNB for Trillioner token is shown as being 0:



1 BNB = 0 TLC

Figure 3 - Wrong swap quotation

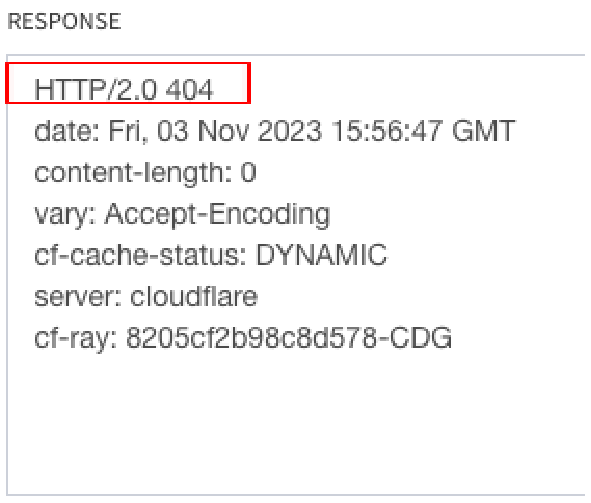
Analyzing the traffic reveal that the API call for the quote responded with 404 (meaning the ressource do not exist):



```
REQUEST

GET https://api.1inch.io/v5.0/56/quote?fromTokenAddress=0xba2ae424d960c26247dd6c32edc70b295c744c43&toTokenAddress=0xbb
accept: application/json, text/plain, */*
accept-encoding: gzip
cookie: __cf_bm=9.J31eoVAW_n6XqYUCAem0X8_tKrqB.kimS59Oj976E-1699027001-0-ASjoeomyswl6a41I5sh4cFgaAHDWsGT001uew
user-agent: okhttp/4.9.2
if-modified-since: Fri, 03 Nov 2023 15:56:45 GMT
```

Figure 4 - Quote request



```
RESPONSE

HTTP/2.0 404
date: Fri, 03 Nov 2023 15:56:47 GMT
content-length: 0
vary: Accept-Encoding
cf-cache-status: DYNAMIC
server: cloudflare
cf-ray: 8205cf2b98c8d578-CDG
```

Figure 5 - Quote response

Based on 1inch documentation, the api endpoint used is deprecated (See https://docs.1inch.io/docs/aggregation-protocol/api/swagger)

Remediation

To continue using the Swap API from 1inch, migrate to the 1inch Developer Portal.

Reference:
https://portal.1inch.dev/documentation/swap/swagger?protocol=BSC

[NC-01] Weak password policy

Severity	Remediation effort
🟦 Informational	🟩 Easy

Description

Application does not enforce using strong passwords. During the tests, it was possible to set the password “12345678”. Lack of a strong password policy increases risk of unauthorized access to an account (see [H-01] Brute-force login mechanism).

Technical details

At this moment the application requires only a minimum password length of 8 characters.

Remediation

It is recommended to implement the requirements regarding password complexity, in particular:

a) Enforcing a minimum password length of at least 12 characters.

b) Include password strength meter to help users create a more complex password and block common and previously breached passwords.

More informations:

- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials

[NC-02] Misleading swap functionality

Severity	Remediation effort
🟦 Informational	🟩 Easy

Description

The swap functionality is designed exclusively for Binance Smart Chain (BSC) tokens, but this crucial restriction is not documented. This omission could potentially mislead users, particularly given the wallet's multi-chain capabilities.

Technical details

This information was found analyzing the traffic. The API call for the swap quote exclusively employs "56" as the chain ID parameter, which signifies that the swap is exclusively intended for the BSC chain.

Remediation

To address this issue, it is advisable to provide clear warnings to users and properly document this behavior, ensuring that they are aware of the restriction regarding BSC tokens and the potential consequences of attempting swaps on other chains.

[NC-03] Potential leak of private keys

Severity	Remediation effort
🟦 Informational	🟩 Easy

Description

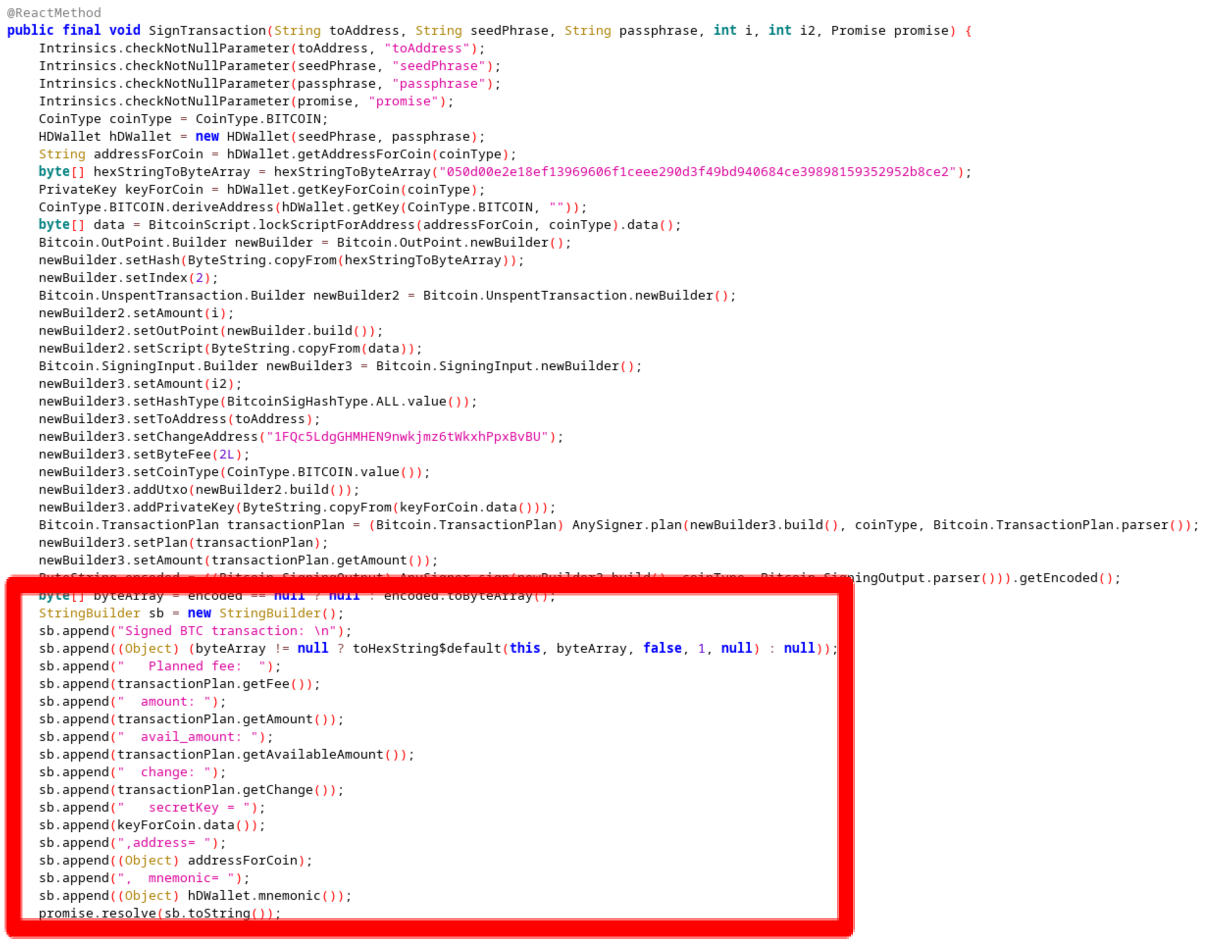
In *com.qfx.trillionerwallet/WalletModule.java*, the SignTransaction function , used to sign BTC transactions, currently returns the private key within its promise result, which contradicts its intended purpose of solely signing the transaction. The expected output should contain only the signed transaction, without exposing the private key.

Risk

The common practice for handling a promise result, such as logging, poses a potential risk of inadvertently leaking private keys. Due to the absence of specific details on the source code, the risk is classified as informational. Nevertheless, be mindful that the impact could be significantly higher.

Technical details

Upon reverse engineering the application using JADX, we can identify the code responsible for signing Bitcoin transactions (refer to the screenshot). This function signs a transaction while generating a result string that includes the signed transaction, as well as sensitive data such as the "secret key" and "mnemonic."



```java
@ReactMethod
public final void SignTransaction(String toAddress, String seedPhrase, String passphrase, int i, int i2, Promise promise) {
    Intrinsics.checkNotNullParameter(toAddress, "toAddress");
    Intrinsics.checkNotNullParameter(seedPhrase, "seedPhrase");
    Intrinsics.checkNotNullParameter(passphrase, "passphrase");
    Intrinsics.checkNotNullParameter(promise, "promise");
    CoinType coinType = CoinType.BITCOIN;
    HDWallet hDWallet = new HDWallet(seedPhrase, passphrase);
    String addressForCoin = hDWallet.getAddressForCoin(coinType);
    byte[] hexStringToByteArray = hexStringToByteArray("050d00e2e18ef13969606f1ceee290d3f49bd940684ce39898159352952b8ce2");
    PrivateKey keyForCoin = hDWallet.getKeyForCoin(coinType);
    CoinType.BITCOIN.deriveAddress(hDWallet.getKey(CoinType.BITCOIN, ""));
    byte[] data = BitcoinScript.lockScriptForAddress(addressForCoin, coinType).data();
    Bitcoin.OutPoint.Builder newBuilder = Bitcoin.OutPoint.newBuilder();
    newBuilder.setHash(ByteString.copyFrom(hexStringToByteArray));
    newBuilder.setIndex(2);
    Bitcoin.UnspentTransaction.Builder newBuilder2 = Bitcoin.UnspentTransaction.newBuilder();
    newBuilder2.setAmount(i);
    newBuilder2.setOutPoint(newBuilder.build());
    newBuilder2.setScript(ByteString.copyFrom(data));
    Bitcoin.SigningInput.Builder newBuilder3 = Bitcoin.SigningInput.newBuilder();
    newBuilder3.setAmount(i2);
    newBuilder3.setHashType(BitcoinSigHashType.ALL.value());
    newBuilder3.setToAddress(toAddress);
    newBuilder3.setChangeAddress("1FQc5LdgGHMHEN9nwkjmz6tWkxhPpxBvBU");
    newBuilder3.setByteFee(2L);
    newBuilder3.setCoinType(CoinType.BITCOIN.value());
    newBuilder3.addUtxo(newBuilder2.build());
    newBuilder3.addPrivateKey(ByteString.copyFrom(keyForCoin.data()));
    Bitcoin.TransactionPlan transactionPlan = (Bitcoin.TransactionPlan) AnySigner.plan(newBuilder3.build(), coinType, Bitcoin.TransactionPlan.parser());
    newBuilder3.setPlan(transactionPlan);
    newBuilder3.setAmount(transactionPlan.getAmount());
    ByteString encoded = ((Bitcoin.SigningOutput) AnySigner.sign(newBuilder3.build(), coinType, Bitcoin.SigningOutput.parser())).getEncoded();
    byte[] byteArray = encoded == null ? null : encoded.toByteArray();
    StringBuilder sb = new StringBuilder();
    sb.append("Signed BTC transaction: \n");
    sb.append((Object) (byteArray != null ? toHexString$default(this, byteArray, false, 1, null) : null));
    sb.append("   Planned fee:  ");
    sb.append(transactionPlan.getFee());
    sb.append("  amount: ");
    sb.append(transactionPlan.getAmount());
    sb.append("  avail_amount: ");
    sb.append(transactionPlan.getAvailableAmount());
    sb.append("  change: ");
    sb.append(transactionPlan.getChange());
    sb.append("   secretKey = ");
    sb.append(keyForCoin.data());
    sb.append(",address= ");
    sb.append((Object) addressForCoin);
    sb.append(",  mnemonic= ");
    sb.append((Object) hDWallet.mnemonic());
    promise.resolve(sb.toString());
```

Figure 6 - SignTransaction function returns secret informations

Remediation

To address this issue, it is advisable not to include the secret key and mnemonic in the result. Alternatively, you can encrypt this sensitive information before transmission.

[NC-04] Missing rooted device detection

Severity	Remediation effort
🟦 Informational	🟧 Moderate

Description

Tested application does not verify whether the device on which the app is running was rooted. Good security practices related to Android environment suggest to verify whether such device was rooted, and at least notify users about that fact.

Remediation

Application should verify whether it is running on a secure (not rooted) device, and at least notify the user that unsecure device is used.

The freely available rootbeer library16 could be considered for the purpose of alerting users on rooted devices.

[NC-05] Certificate transparency not implemented

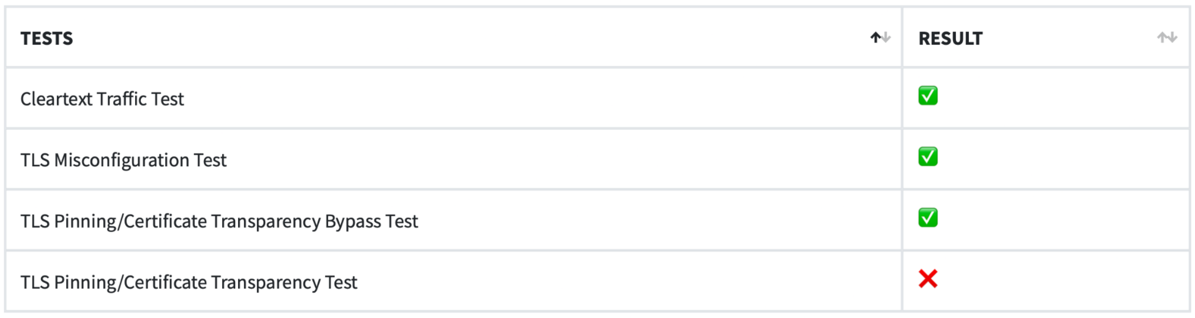
Severity	Remediation effort
🟦 Informational	🟧 Moderate

Description

The app's failure in the TLS Pinning/Certificate Transparency test could potentially expose the app to man-in-the-middle (MITM) attacks, where an attacker intercepts and decrypts the communication between the app and its servers, compromising data security.

Technical details

Test conducted with MobSF.



TESTS	RESULT
Cleartext Traffic Test	✅
TLS Misconfiguration Test	✅
TLS Pinning/Certificate Transparency Bypass Test	✅
TLS Pinning/Certificate Transparency Test	❌

Figure 7 - MobSF TLS/SSL Security Test

Remediation

Consider implementing Certificate Transparency as it is well-suited for mobile applications.

Reference: https://github.com/appmattus/certificatetransparency

Appendix - Document Informations

Document Revision			
Version	Date	Details	Author
1.0	04/11/2023	Initial report	Maleak Teouri

Contact information	
Name	Maleak Teouri
Phone	+33 6 51 23 16 97
Email	contact@mteouri.com