FIGI

FINANCIAL INCLUSION
GLOBAL INITIATIVE

SECURITY, INFRASTRUCTURE AND TRUST WORKING GROUP

Security audit of various DFS applications

REPORT OF THE SECURITY WORKSTREAM

Security, Infrastructure and Trust Working Group

Security audit of various DFS applications

DISCLAIMER

The Financial Inclusion Global Initiative (FIGI) is a three-year program implemented in partnership by the World Bank Group (WBG), the Committee on Payments and Market Infrastructures (CPMI), and the International Telecommunication Union (ITU), funded by the Bill & Melinda Gates Foundation (BMGF) to facilitate the implementation of country-led reforms to attain national financial inclusion targets, and ultimately the global 'Universal Financial Access 2020' goal. FIGI funds initiatives in three countries-China, Egypt and Mexico; supports working groups to address three distinct challenges for reaching universal financial access:

(1) The Electronic Payment Acceptance Working Group (led by the WBG),
(2) The Digital ID for Financial Services Working Group (led by the WBG), and
(3) The Security, Infrastructure and Trust Working Group (led by the ITU).

FIGI hosts three annual symposia to assemble national authorities, the private sector, and other relevant stakeholders to share emerging insights from the Working Groups and country level implementation.

This report is a product of the FIGI Security, Infrastructure and Trust Working Group, led by the International Telecommunication Union. The findings, interpretations, and conclusions expressed in this work do not necessarily reflect the views of the Financial Inclusion Global Initiative partners including the Committee on Payments and Market Infrastructures, the Bill & Melinda Gates Foundation, the International Telecommunication Union, or the World Bank (including its Board of Executive Directors or the governments they represent). The mention of specific companies, or of certain manufacturers' products does not imply that they are endorsed nor recommended by ITU in preference to others of a similar nature that are not mentioned. Errors and omissions excepted, the names of proprietary products are distinguished by initial capital letters. The FIGI partners do not guarantee the accuracy of the data included in this work. The boundaries, colours, denominations, and other information shown on any map in this work do not imply any judgment on the part of the FIGI partners concerning the legal status of any country, territory, city or area or of its authorities or the endorsement or acceptance of such boundaries.

About this report

This report was written by Sébastien Mathieu and Philippe Oechslin of Objectif Sécurité. The author would also like to thank the FIGI Security Infrastructure and Trust working group members for their contributions and comments.

If you would like to provide any additional information, please contact Vijay Mauree at tsbfigisit@itu.int

Contents

Executive Summary

The main objective of this report is to present the findings of the security audit of a few mobile digital financial services (DFS) applications operating on Android mobile operating system and elaborate a systematic methodology for carrying out the security audit.

The security audit methodology is based on 18 tests and are categorised according to seven of the well-known categories of the OWASP Mobile top 10 security risks[1]. Three mobile DFS applications were tested: Two DFS applications from providers in Africa and one DFS application from Europe. The results of security audit of these applications have been anonymised in the report and the three DFS apps are referred to as App1, App2 and App3.

An overview of the results is shown in the table below:

OWASP mobile top 10	Test	App1	App2	App3
M1: Improper Platform Usage	T1.1 Android:allowBackup	✓	✓	✓
	T1.2 Android:debuggable	✓	✓	✓
	T1.3 Android:installLocation	✓	✓	✓
	T1.4 Dangerous permissions	✓	✓	✓
M2: Insecure Data Storage	T2.1 Android.permission. WRITE_EXTERNAL_STORAGE	×	✓	✓
	T2.2 Disabling screenshots	✓	✓	×
M3: Insecure Communication	T3.1 Application should only use HTTPS connections	✓	✓	✓
	T3.2 Application should detect Machine-in-the-Middle attacks with untrusted certificates	✓	✓	✓
	T3.3 Application should detect Machine-in-the-Middle attacks with trusted certificates	✓	✓	×
	T3.4 App manifest should not allow cleartext traffic	✓	×	×
M4: Insecure Authentication	T4.1 Authentication required before accessing sensitive information	✓	×	×
	T4.2 The application should have an inactivity timeout	✓	✓	✓
	T4.3 If a fingerprint is added, existing authentication with fingerprints should be disabled	✓	✓	✓
	T4.4 Sensitive requests cannot be replayed	✓	×	✓
M5: Insufficient Cryptography	T5.1 The app should not use unsafe crypto primitives	×	×	×
	T5.2 The HTTPS connections should be configured according to best practices	✓	✓	✓
	T5.3 The app should encrypt sensitive data that is sent over HTTPS	×	✓	✓
M8: Code Tampering	T8.1 The application should refuse to run on a rooted device	×	✓	✓
M9: Reverse Engineering	T9.1 The code of the app should be obfuscated	✓	✓	✓

The 18 security tests defined in the methodology for the security audit in the report has also been mapped against the security best practices of the DFS Security Assurance Framework report, illustrating how the security tests can verify the adherence to the best practices by the application being tested (see Chapter 4 of the report). This methodology will be used in the DFS Security Lab for security of DFS applications based on Android platform. The DFS Security Lab has been set up by the ITU as part of the activities of the Security, Infrastructure and Trust (SIT) Working Group under the Financial Inclusion Global Initiative (FIGI).

Abbreviations

CA	Certificate Authority
DES	Data Encryption Standard
DFS	Digital Financial Services
ECB	Electronic Code Book
HTTPS	Hyper Text Transfer Protocol
MD	Message Digest
MITM	Man in the Middle
OWASP	Open Web Application Security Project
PIN	Personal Identification Number
PUK	Personal Unlock Key
RC	Rivest Cipher
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security

Security audit of Android DFS applications

1 ABOUT THE APPS

The three DFS applications analysed were selected as follows: Two DFS applications from providers in Africa and one DFS application from Europe. The results of security audit of these applications have been anonymised in the report and the three DFS apps are referred to as App1, App2 and App3.

1.1 App1

A European based mobile payment app, App1 links the user's credit card and bank account. It can be used to send, request and receive money. The app can also be used to make online payments by scanning QR codes and can be used to make cashless payments at stores, restaurants, pay for parking tickets using QR codes or merchant beacons. A user requires a mobile number and a credit card or bank account details to register.

1.2 App2

App2 is provided by a mobile network operator that provides digital financial services in areas in which they operate across Africa. The innovative mobile financial service application makes it possible for users to send money locally and internationally, pay for goods and services, and transact from anywhere in the world, make transfers between the mobile wallet and user's bank account. To register the app requires a mobile number with the operator. The app users do not need to have a bank account.

1.3 App3

App3 is also provided a mobile operator and in several countries across Africa and Asia. The app makes it possible for users to send money to contacts, pay for goods and services, the app also transfers between the mobile wallet and bank account. To register the app requires a mobile number with the operator. The app users do not need to have a bank account.

2 TESTING METHOD

The goal of these tests is to give a standardised score of the security level of smart phone apps for Digital Financial Services. This is achieved by installing the app on a test phone and analysing its security features with a set of testing tools. The tests have been chosen such that they can be carried out with open-source tools and with reasonable effort.
The tests are organised according to the OWASP mobile top 10 list. The Open Web Application Security Project[1] (OWASP) is a non-profit foundation that works to improve the security of software. One of their projects is the OWASP mobile top 10[2] which list the following risks as most important:

a) M1 Improper Platform Usage
b) M2 Insecure Data Storage
c) M3 Insecure Communication
d) M4 Insecure Authentication
e) M5 Insufficient Cryptography

f) M6 Insecure Authorization
g) M7 Client Code Quality
h) M8 Code Tampering
i) M9 Reverse Engineering
j) M10 Extraneous Functionality

Categories M6, M7 and M10 are out of the scope of our tests as they would require access to the source code of the application or reverse engineering the logic of the application.
The following set of 18 tests has been selected based on their pertinence and of the feasibility of the test:

2.1 M1 Improper Platform Usage

These tests are done by analysing the manifest of the application. The following issues are verified:

a) **T1.1 Android:allowBackup**[3]:
   This setting should be set to false, which is not the default value.
   If this attribute is set to false, no backup or restore of the application will ever be performed, even by a full-system backup that would otherwise cause all application data to be saved
b) **T1.2 Android:debuggable**[4]:
   This setting should be set to false, which is the default value.
   If an application is flagged as debuggable, intruders can inject their own code to execute it in the context of the vulnerable application process.
c) **T1.3 Android:installLocation**[5]:
   This should be set to internalOnly or unset, which is the default value.
   If this parameter is set to auto or preferExternal, the application may be installed on a removable memory card.
   By gaining access to the memory card, an attacker could be able to tamper with the application or to extract sensitive information.
   Note that even if the application fails this test, it does not necessarily mean that the application will be installed on the removable card.
d) **T1.4 Dangerous permissions**:
   The application should not require dangerous permissions without a valid reason.
   Android apps must explicitly ask for permissions for many types of operations. Some of these permissions are labelled "dangerous" by Android. The app must explicitly ask the user to grant dangerous permissions with a dialog (e.g., Allow App to make phone calls?). There may be valid reasons for requesting dangerous permissions. For example, a DFS application that needs to scan QR codes for making payments will need the permission to use the camera.
   Note that permissions regarding storage of data are considered in the next section.
   An app that requires dangerous permissions could abuse the permissions to attack the user. For example, it could dial premium rated phone numbers if dialling permission were granted.

2.2 M2 Insecure Data Storage

These tests are also done by analysing the manifest of the application and by running app on a phone. Following issues are verified:

a) **T2.1 Android.permission. WRITE_EXTERNAL_STORAGE**:
   The app should not require this permission without a valid reason.
   This permission allows the app to read and write data on a memory card inserted in the phone. If the application needs to store substantial amounts of non-sensitive information, this would justify writing to external storage.
   By gaining access to the memory card, an attacker could be able to tamper with the application or to extract sensitive information.
   Note that even if the application fails this test, it does not necessarily mean that the application will write sensitive data to external storage.
b) **T2.2 Disabling screenshots**:
   The app should disable screenshots while it is running and only show a blank image when in the task switcher.
   This is a standard behaviour for secure applications and can be achieved with an application parameter called FLAG_SECURE[6]. This can be tested by running the application and a) trying to make a screenshot, b) switching between apps and observer the thumbnail of the application.
   Without this setting, a malicious application could potentially steal sensitive information from the screen of an application.

2.3 M3 Insecure Communication

a) **T3.1 Application should only use HTTPS connections**:
   When running the traffic of the app through an audit machine and observing the packets, only HTTPS traffic should be observed for the application.

HTTPS traffic is encrypted. While they are other ways to encrypt traffic, HTTPS is the standard way for communication between apps and servers. If data is transferred over HTTP or other non-encrypted protocols, then it could easily be intercepted or even modified by an attacker.

b) **T3.2 Application should detect Machine-in-the-Middle attacks with untrusted Certificates**:
When running the traffic through a machine-in-the-middle (MITM) proxy that does not own a trusted certificate for the server of the app, the app should refuse the connection.
MITM proxies can be used to intercept HTTPS traffic, decrypt it for inspection and modification and the re-encrypt it before sending it off to the intended server. A typical attacker does not own valid certificates for the destination server; thus the app should detect that the certificate of the proxy is not signed by a trusted authority. If the app does not check the validity of the certificate, an attacker can intercept and modify the traffic.

c) **T3.3 Application should detect Machine-in-the-Middle attacks with trusted certificates**:
When running the traffic through a machine-in-the-middle (MITM) proxy that uses a certificate signed by a CA that is trusted by the smart phone, the app should refuse the connection.
Different situation can arise where the operator of the proxy is able to generate certificates that are trusted by the phone. The operator can be a CA operator (e.g., a government), the operator may be a company that has installed its root certificate on the phones of the company, or the root certificate may have been installed by hand by the user or an attacker. The application can protect itself against this type of attack by doing a root pinning. This means that the app knows which CA is expected to sign the server certificate and it will refuse certificates signed by other CAs, even if these CAs are trusted. Executing this test usually requires rooting of the phone to be able to install a root certificate.
If the application does not apply certificate pinning, then traffic could be intercepted by governments or by attackers having succeeded in hacking one of the many trusted root CAs.

d) **T3.4 App manifest should not allow clear text traffic**:
Using clear text traffic is disabled by default on Android 8.1 or higher. The app manifest should not contain settings that override this default. These can be the android:usesCleartextTraffic setting for the application or clear textTrafficPermitted in the network security configuration.
When clear text traffic is disabled, the application and other components it uses (e.g., the media player) will refuse to use clear text traffic. Clear text traffic can easily be eavesdropped and manipulated by attackers.
Note that even if the application fails this test, it does not necessarily mean that the application will send or receive clear text traffic.

2.4 M4 Insecure Authentication

The following tests are made by running the application on a phone and observing its behaviour.

a) **T4.1 Authentication required before accessing sensitive information**:
The app should request a password, a PIN code, or a fingerprint before giving access to sensitive information or functionality (e.g., balances and payments).
This can be tested by using the application on the phone.
The impact of not authenticating the user every time is that if the phone is stolen or lent while unlocked, an attacker could access sensitive data or functionality.

b) **T4.2 The application should have an inactivity timeout**:
This can be tested by leaving the application open for a while and observing whether it locks itself automatically.
If there is no timeout, or if it is too long, the risk is that if the phone is stolen or lent while unlocked, an attacker could access sensitive data or functionality.

b) **T4.3 If a fingerprint is added, authentication with fingerprints should be disabled**:
When a new fingerprint is registered on the phone, the app should disable authenticate by fingerprint until the user has provide the PIN or password for the application.
The risk is that an attacker could succeed in registering his own fingerprint on the phone and the access the apps that are protected by fingerprints.

c) **T4.4 It should not be possible to replay intercepted requests**:
Replaying a request (e.g., a money transfer) that was captured by a man-in-the-middle proxy should not result in the same request being executed twice.

The risk is that an attacker intercepting a request for a money transfer could replay it to steal money from the victim.

2.5 M5 Insufficient Cryptography

a) **T5.1 The app should not use unsafe crypto primitives:**
Algorithms like MD5, SHA-1, RC4, DES, 3DES, Blowfish, ECB mode for block ciphers, non-cryptographic random generators are known to be weak and should not be used by the application[7]. This can be tested by analysing the binary of the application to see if it makes calls to these unsafe algorithms.
If sensitive information were handled or dependent on these algorithms, the there is a risk that an attacker could eavesdrop or manipulate that information. The fact that these algorithms are used does not necessarily mean that they are used for sensitive operations. Still, it is a best practice to not use these algorithms to create any doubt.
Note that even if the application fails this test, it does not necessarily mean that the application uses unsafe crypto primitives for sensitive data.

b) **T5.2 The HTTPS connections should be configured according to best practices**:
By observing the network traffic of the app, the servers to which it talks can be identified. The HTTPS configuration of these servers can be tested using a tool like Qualys SSL Labs[8]. The overall rating should be B or more.
If HTTPS is not correctly configured, then some eavesdropping or manipulation attacks are possible.

c) **T5.3 The app should encrypt sensitive data that is sent over HTTPS**:
This can be tested by intercepting the traffic with an MITM proxy (see tests in M3). Note that if the app uses certificate pinning, it is necessary to disable this protection to intercept the traffic. This is not always possible.
If data is not encrypted by the application itself, then a MITM can eavesdrop or modify the data.

2.6 M8 Code Tampering

T8.1 The application should refuse to run on a rooted device:
When installed on a rooted android phone, the application should refuse to run.
Several security mechanisms can be disabled on rooted phone. This would allow an attacker to tamper the code or the data of the application to commit fraud.
If the application accepts to run on a rooted device, then it should at least apply the following three security controls: Obfuscation of the code (T9.1), apply certificate pinning to prevent interception of communication with trusted certificates (T3.3) and sensitive information should be encrypted by the application, even is transmitted over HTTPS (T5.3).

2.7 M9 Reverse Engineering

T9.1 The code of the app should be obfuscated:
Several tools can be used to analyse the binary of the app and detect if it has been obfuscated. Alternatively, the code can tentatively be decompiled with a decompiler. If it succeeds, the decompiled code can be analysed to see if it is intelligible.
Obfuscating the code makes it much more difficult to understand and analyse its logic and algorithms.

3 RESULTS

3.1 App1

App1 can be used to send money between users or to pay without cash in stores or at vending machines. Users are identified by their phone number. Knowing another user's phone number is all that is needed to send money. Accounts are typically backed by a bank account. It is also possible to have a prepaid account that is independent of a bank account.

3.1.1 M1: Improper Platform Usage

√ T1.1 Android:allowBackup is set to false in the manifest.
√ T1.2 Android:debuggable is not defined in the manifest.

√ T1.3 Android:installLocation is not defined in the manifest.
√ T1.4 We did not find inappropriate Android permissions in the manifest.

3.1.2 M2: Insecure Data Storage

x T2.1 The application requires the "android.permission.WRITE_EXTERNAL_STORAGE" permission. Note that this does not imply that the app writes data on external storage and, if it did, that this data is sensible.
√ T2.2 While the app is running, screenshots are disabled.

3.1.3 M3: Insecure Communication

√ T3.1 Only HTTPS connections are used.
√ T3.2 The app refused to establish an HTTPS connection to a proxy with an untrusted certificate.
√ T3.3The app refused to establish HTTPS connection to a proxy with a trusted certificate. This shows that certificate pinning is in use.
√ T3.4 The application defines a custom network security configuration in its manifest. This configuration disables clear text traffic:

```
<network-security-config>
    <base-config clear textTrafficPermitted="-
false">
  ...
 </base-config>
</network-security-config>
```

3.1.4 M4: Insecure Authentication

√ T4.1 Every time the app is started, the app requires a PIN or a fingerprint to authenticate.
√ T4.2 The application implements an inactivity timeout. After a period of inactivity, the application logs out.
√ T4.3 If a fingerprint is added, the application disables authentication with fingerprints.
√ T4.4 Money send requests cannot be successfully replayed. The server responds with a "409 Conflict" error message and does not process the money send request.

3.1.5 M5: Insufficient Cryptography

x T5.1 The application uses the weak MD5 and SHA-1 hashing algorithms as well as the weak ECB mode of encryption.

MD5 in file com/appdynamics/eumagent/runtime/p000private/ae.java:
MessageDigest instance = MessageDigest.getInstance("MD5");
SHA-1 in file com/App1/android/Security/SecCore/b/a.java:
MessageDigest instance = MessageDigest.getInstance("SHA-1");
ECB in file com/App1/android/Security/SecCore/b/a.java:
Cipher instance = Cipher.getInstance("AES/ECB/NoPadding");

√ T5.2 By intercepting the applications HTTPS requests with Burp Proxy, the server to which the client connects to could be identified. The TLS configuration of the server was assessed using Qualys SSL Labs[9]. It had an overall rating is A+.
x T5.3 By intercepting the applications HTTPS requests with Burp Proxy, the client requests are signed. However, the amount of money transferred and the first name, last name and phone number of the users participating in the transfer are in clear text.

3.1.6 M8: Code Tampering

√ T8.1 We were able to install and run the app on a rooted device.

3.1.7 M9: Reverse Engineering

√ T9.1 The app code has been obfuscated as shown in Figure 1.

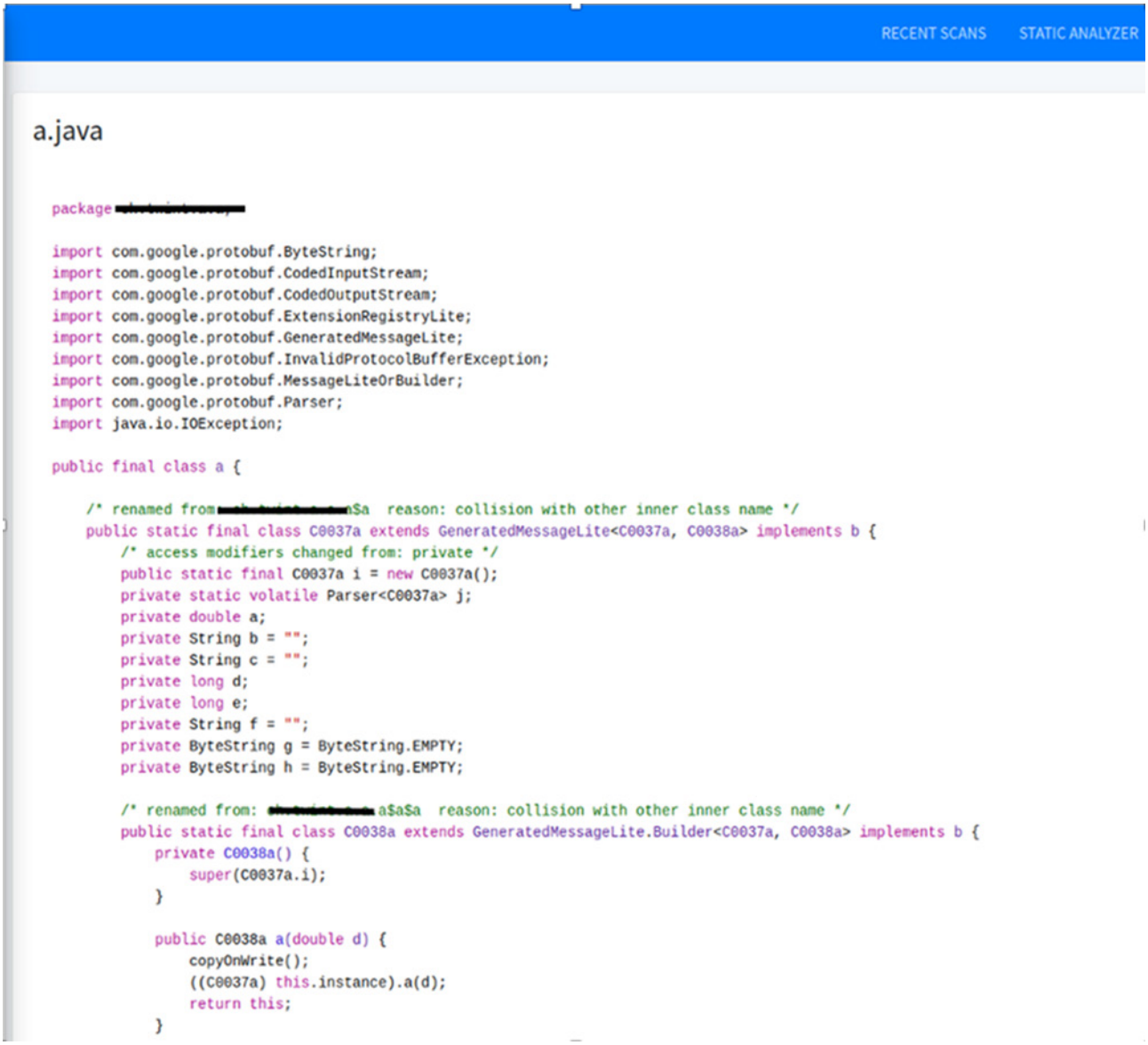
3.2 App2

App2 is used for mobile money transfer, payment, and micro-financing service. App2 is not backed by a bank account. Money can be deposited and withdrawn from accounts through different agents like airtime resellers or retail outlets.

3.2.1 M1: Improper Platform Usage

√ T1.1 Android:allowBackup is set to false in the manifest.
√ T1.2 Android:debuggable is not defined in the manifest.
√ T1.3 Android:installLocation is not defined in the manifest.
√ T1.4 We did not find inappropriate Android permissions in the manifest.

Figure 1 – Names of files, classes and variables have been replaced, making the code more difficult to understand



3.2.2 M2: Insecure Data Storage

√ T2.1 The applications require the "android.permission.WRITE_EXTERNAL_STORAGE" permission. Note that this does not imply that the app actually writes data on external storage and, if it did, that this data is sensible.

√ T2.2 While the app is running, screenshots are disabled.

3.2.3 M3: Insecure Communication

√ T3.1 Only HTTPS connections are used.

√ T3.2 The app refused to establish an HTTPS connection to a proxy with an untrusted certificate

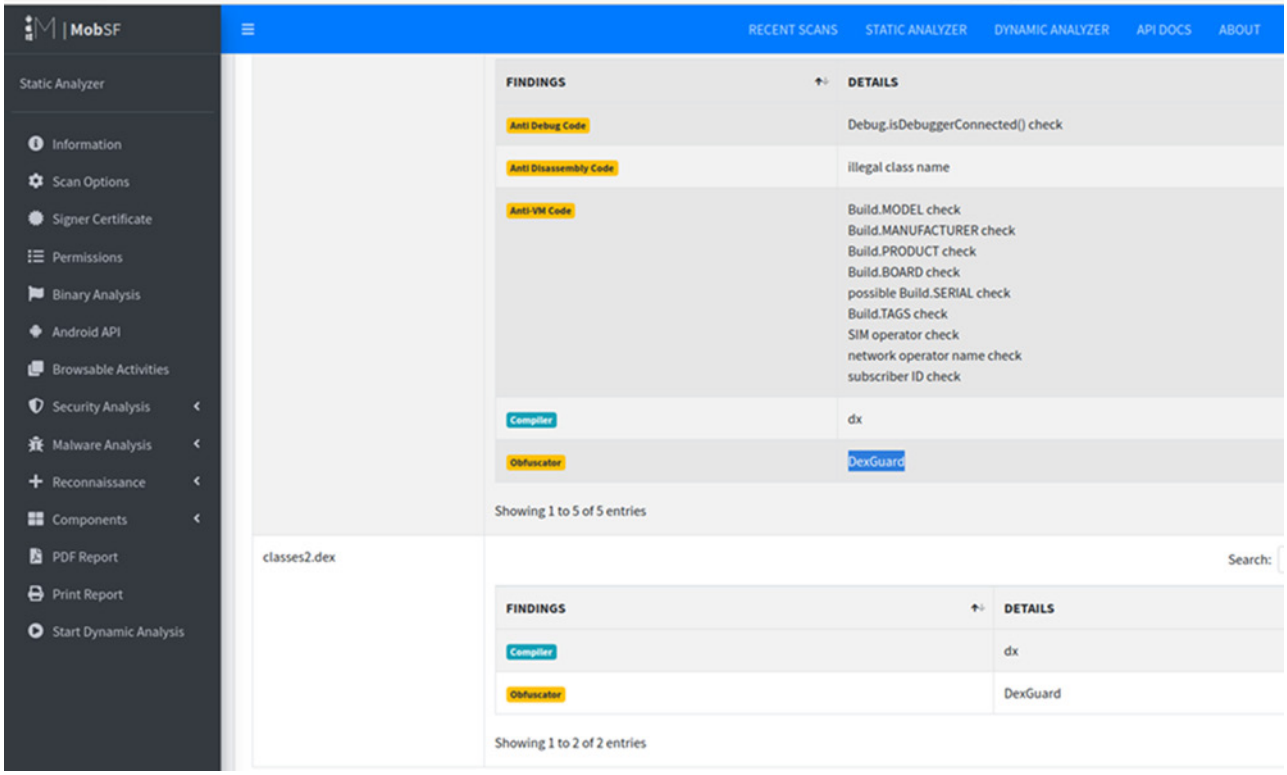
√ T3.3The app refused to establish HTTPS connection to a proxy with a trusted certificate. This shows that certificate pinning is in use.

x T3.4 Android:usesClear textTraffic is set to true in the manifest.

3.2.4 M4: Insecure Authentication

x T4.1 The application does not require a PIN or fingerprint every time it is started. Thus, an intruder stealing an unlocked device can run the applica-

Figure 2 – App2 is protected with DexGuard



tion. This allows an attacker to see the PUK of the phone and the balance of the account. The PIN is however required to make money transactions.

√ T4.2 The PIN is required for each money transfer. This is even safer than a timeout.

√ T4.3 If a fingerprint is added, the application disables authentication with fingerprints.

x T4.4 Sensitive requests like money transfers can be replayed by an MITM proxy.

3.2.5 M5: Insufficient Cryptography

x T5.1 The application uses the weak SHA-1 hashing algorithm as well as the weak default random number generator.
SHA-1 in file o/C1668.java:
MessageDigest instance = MessageDigest.getInstance("SHA-1");

x Random generator in file o/C1783.java:
this(juVar, d, new Random());

√ T5.2 By intercepting the applications HTTPS requests with Burp Proxy, the server to which the application connects was identified.
The TLS configuration of the server was tested using Qualys SSL Labs[10]. Its overall rating is A+.

3.2.6 M8: Code Tampering

√ T8.1 The app does not run on a rooted Android device.

3.2.7 M9: Reverse Engineering

√ T9.1 The app code has been obfuscated. The app code has been obfuscated by DexGuard[11] as shown in Figure 2.

3.3 App3

App3 is a payment app that can be used to pay utilities, to transfer money or to shop online. It can either be linked to a bank account or to a digital wallet registered to a home number.

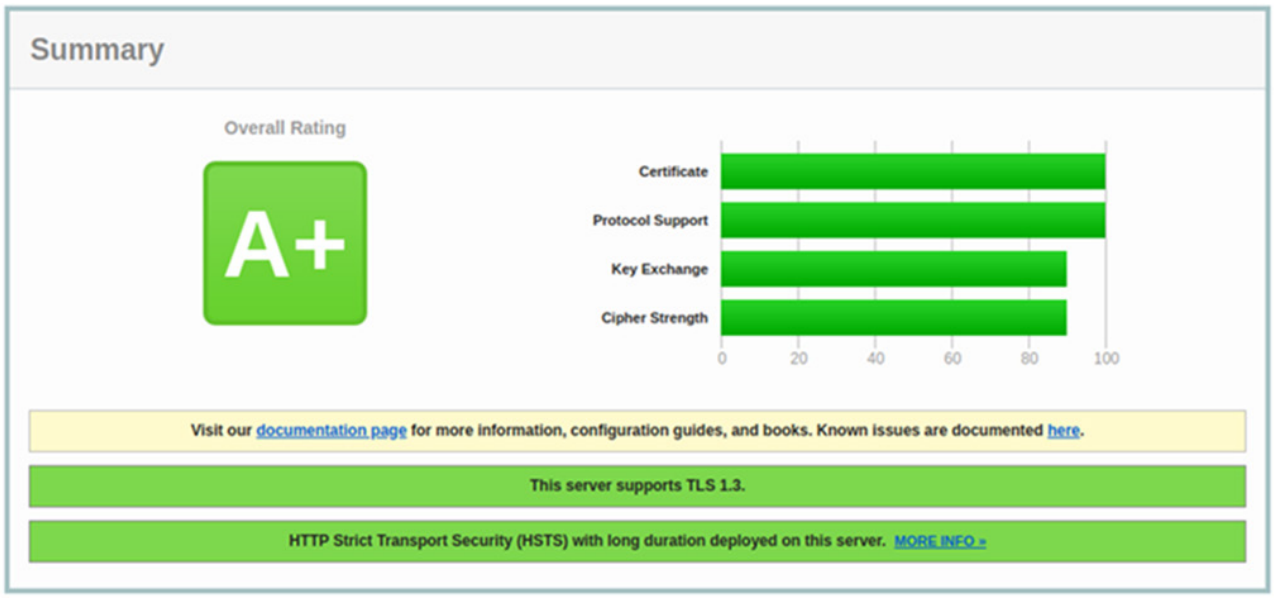
3.3.1 M1: Improper Platform Usage

√ T1.1 Android:allowBackup is set to false in the manifest.

√ T1.2 Android:debuggable is not defined in the manifest.

√ T1.3 Android:installLocation is not defined in the manifest.

Figure 3 – SSL configuration assessment of app3's server



√ T1.4 We did not find inappropriate Android permissions in the manifest.

3.3.2 M2: Insecure Data Storage

√ T2.1 The application does not require the "android.permission.WRITE_EXTERNAL_ STORAGE" permission.

x T2.2 While the app is running, screenshot is not disabled. Moreover, the background screenshots from the recent-tasks history are not blurred.

3.3.3 M3: Insecure Communication

√ T3.1 Only HTTPS connections are used.

√ T3.2 The app refused to establish an HTTPS connection to a proxy with an untrusted certificate.

x T3.3 The app accepts to establish an HTTPS connection to a proxy with a trusted certificate. This shows that certificate pinning is not in use.

x T3.4 Android:usesClear textTraffic is set to true in the manifest.

3.3.4 M4: Insecure Authentication

x T4.1 The application does not require a PIN or fingerprint every time it is started. Thus, an intruder stealing an unlocked device can run the application. This allows an attacker to see the balance of the account. The PIN is however required to make money transactions.

√ T4.2 The PIN is required for each money transfer. This is even safer than a timeout.

√ T4.3 If a fingerprint is added, the application disables authentication with fingerprints.

x T4.4 Sensitive requests like money transfers can be replayed by an MITM proxy.

3.3.5 M5: Insufficient Cryptography

x T5.1 The application uses the weak MD5 and SHA-1 hashing algorithms as well as the weak default random number generator.
MD5 in file com/appsflyer/internal/ai.java:
MessageDigest instance = MessageDigest.getInstance("MD5");
SHA-1 in file u/b/a/a/o/b/j.java:
MessageDigest instance = MessageDigest.getInstance("SHA-1");
Random generator in file c/g/a/c/s.java:
Random random = new Random();

√ T5.2 By intercepting the applications HTTPS requests with Burp Proxy, the server to which the application connects to was identified.
The TLS configuration of the identified server was tested using Qualys SSL Labs[12]. Its overall rating is A+.

√ T5.3 By intercepting the application's HTTPS requests with Burp Proxy, it was observed that the body of some sensitive requests is encrypted.
However, some responses with sensitive data like the current balance are neither encrypted nor

authenticated, Thus they can be modified to carry out malicious attacks.
The server response **was tampered during testing and user's current balance showing on the app was modified.**

3.3.6 M8: Code Tampering

√ T8.1 The app does not run on a rooted Android device.

3.3.7 M9: Reverse Engineering

√ T9.1 The app code has been obfuscated.

4 CONCLUSIONS

The method excludes analysis the logic of the applications, as this would require reverse engineering of the application code. All three tested applications make use of code obfuscation, which would make reverse engineering particularly challenging.

4.1 Evaluating the results

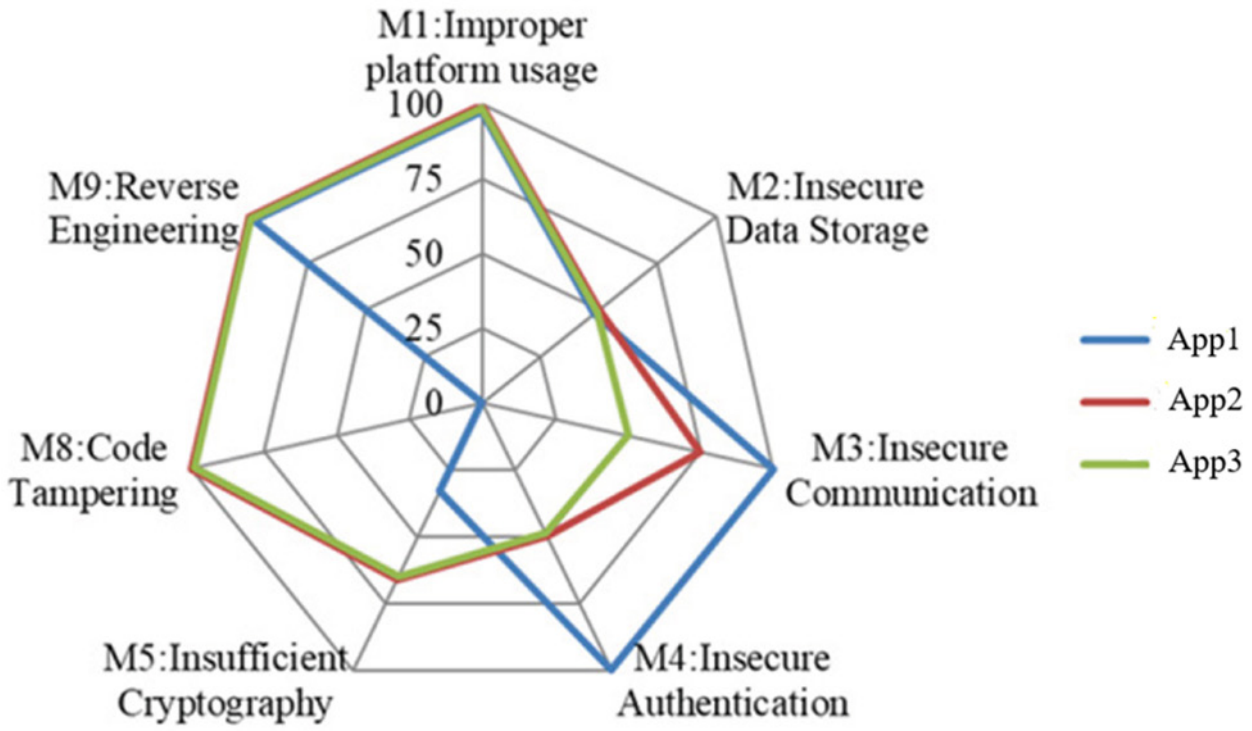
Since we do not analyse the logic of the applications, it is difficult to estimate the impact of a failed test. For example, the detection of insecure cryptographic operations does not necessarily imply that sensitive information will not be encrypted in a safe manner. Another example is the fact that App1 does not encrypt the details of transactions (names and amounts) that are transmitted through HTTPS. Since the application uses certificate pinning, there is a limited chance that the information can be intercepted by an adversary.
Still, all the tests are related to best practices which should be followed by financial applications. The results should thus be read as a standardized evaluation of whether the applications are built according to best practices. Whether an application is vulnerable to a specific attack cannot be deduced from the test results without additional investigation.

4.2 Comparing to the FIGI SIT DFS Security Assurance Framework

The Security, Infrastructure and Trust working group of the Financial Inclusion Global Initiative (FIGI) has established a DFS security assurance framework[13].
In its chapter 9 of the DFS security assurance framework report, a template of five categories of security best practices are provided. The following table maps the 18 tests of the proposed method to the five categories of best practices:

Best practice from DFS Assurance Framework	Corresponding tests
9.1 Device integrity	T1.2 Android:debuggable T1.4 Dangerous permissions T8.1 The application should refuse to run on a rooted device
9.2 Communication Security and Certificate Handling	T3.1 Application should only use HTTPS connections T3.2 Application should detect Machine-in-the-Middle attacks with untrusted certificates T3.3 Application should detect Machine-in-the-Middle attacks with trusted certificates T3.4 App manifest should not allow clear text traffic T5.1 The app should not use unsafe crypto primitives T5.2 The HTTPS connections should be configured according to best practices T5.3 The app should encrypt sensitive data that is sent over HTTPS
9.3 User authentication	T4.1 Authentication required before accessing sensitive information T4.2 The application should have an inactivity timeout T4.3 If a fingerprint is added, authentication with fingerprints should be disabled
9.4 Secure Data Handling	T1.1 Android:allowBackup T1.3 Android:installLocation T2.1 Android.permission.WRITE_EXTERNAL_STORAGE T2.2 Disabling screenshots
9.5 Secure Application Development	T9.1 The code of the app should be obfuscated

Figure 4 – Radar graph of the test results (the radial axis indicates the percentage of tested best practices that were found to be implemented)



4.3 Summary of results

We conclude this report by summarizing the results of the tests in Figure 4. No critical vulnerability was detected during the tests. Nevertheless, two findings were found.

a) No PIN is required to access a Personal Unlock Key (PUK) in App2

b) App3 does not apply an extra encryption of the data exchanged over HTTPS.

Testing of additional applications would yield a larger base to compare with and would also allow to fine-tune the tests.

Endnotes

1 https://owasp.org

2 https://owasp.org/www-project-mobile-top-10/

3 https://developer.android.com/guide/topics/manifest/application-element#allowbackup

4 https://developer.android.com/guide/topics/manifest/application-element#debug

5 https://developer.android.com/guide/topics/manifest/manifest-element#install

6 https://developer.android.com/reference/android/view/WindowManager.LayoutParams#FLAG_SECURE

7 MD5 and SHA-1 are used to protect integrity of data, RC4, DES, 3DES, Blowfish and ECB protect confidentiality while random generators are used to generate keys for protecting integrity, confidentiality or other properties.

8 https://www.ssllabs.com/ssltest/

9 https://www.ssllabs.com/ssltest/

10 https://www.ssllabs.com/ssltest/

11 https://www.guardsquare.com/en/products/dexguard

12 https://www.ssllabs.com/ssltest/

13 https://www.itu.int/en/ITU-T/extcoop/figisymposium/Documents/ITU_SIT_WG_Technical%20report%20on%20Digital%20Financial%20Services%20Security%20Assurance%20Framework_f.pdf

International Telecommunication Union
Place des Nations
CH-1211 Geneva 20
Switzerland