MOBILE APPLICATION SECURITY ASSESSMENT REPORT

StagingNet & Wallet Staging

PACKAGE NAME: com.concordium.id.staging/com.concordium.wallet.staging

APP VERSION: 0.1/0.5.37

OS: iOS, Android

PREPARED FOR: Concordium

DATE: 2021-05-03

NowSecure™

TABLE OF CONTENTS

NowSecure™

DOCUMENT VERSION HISTORY

Version	Date	Description
1.0	2021-05-03	Report delivered to customer
2.0	2021-05-06	Scoring Update to 2.9

1.0 SUMMARY AND RECOMMENDATIONS

1.1 Executive Summary

For this mobile application security assessment, the application was installed and tested on several test devices. The application and test devices were subjected to extensive forensic analysis, network assessment, and security tests with the objective of identifying security weaknesses and sensitive data exposure that could present a risk to the application's provider or users. Forensic analysis of the application revealed the application cached the UI in both versions, this may allow sensitive data to be leaked. The Android app was also found to store sensitive data in app folder databases in plaintext. The iOS app was also found to use the default platform data protections in its configuration. Network analysis revealed that the app does not use the secure cookie flag. Server-side testing revealed that third party endpoints used by the app accept weak versions of the TLS protocol.

In addition to forensic and network testing, the application was also reverse engineered. This activity reveals what additional information or exploitation vectors an attacker could discover. It was found that the app disables free security controls for network security, and binary protections. These protections prevent the use of HTTP, an insecure network protocol, and memory protections. In addition, third party keyboard usage was not disabled in the iOS app. Finally, the iOS app contains UITextField objects that may expose sensitive data to the device keyboard cache. In summary, 3 medium-severity, and 6 low-severity risks were identified during the assessment of the application. Please review the provided recommendations and remediations and take steps as appropriate to strengthen the security posture of the app.

1.2 Background and Objectives

NowSecure was retained by Concordium to perform a Mobile Application Security Assessment on the StagingNet-Wallet Staging application for the iOS, Android platform(s). The objective for this security assessment and risk mitigation project was to evaluate the security and data exposure risks presented by the use of the application and to provide specific recommendations for mitigating these risks.

1.3 Scope of Testing

This section defines the standard scope of work to be expected from a Mobile Application Security Assessment. NowSecure analysts split the assessment into four categories: Device, Network, Backend, and Reverse Engineering.

Device

Analysts install the application on real devices with the target OS (iOS or Android) and conduct a forensic analysis of the device for specific application or data storage vulnerabilities. The scope of device testing includes:

- Application installation.
- Identify sensitive data stored on the device (in plaintext or reversible hashing/encoding).

- Evaluate common areas of storage for the application and its data files.
- Investigate Keychain/Keystore artifacts.
- Database structure and content evaluation.
- Biometric authentication.
- Sensitive data retained in memory.
- Data encryption.

Network

Analysts operate all aspects of the application as a user would and attempt to detect vulnerabilities via network communications. Taking the position of an attacker, analysts will compromise the network in a variety of ways. The scope of network testing includes:

- Identify sensitive data sent over the network (in plaintext or reversible hashing/encoding).
- Evaluate login/logout process.
- Evaluate session management techniques.
- Evaluate the security of the certificate exchange for HTTPS communications.
- Evaluation Multi-factor authentication implementation.

Backend

Analysts conduct reconnaissance and limited exploitation of backend services that the mobile application interacts with. The scope of backend testing includes:

- Evaluating server cipher negotiation strength.
- Evaluate API authorization and rate limiting implementation.
- Evaluate session management techniques.
- Investigate user/token discovery and enumeration efficacy.

Reverse Engineering

Analysts take the role of an attacker with limited knowledge of the application and attempt to reverse engineer the application to discover how the application works, determine if any sensitive data can be found in reversed binary source code, and attempt to manipulate the application. Reverse engineering scope includes:

- Source code obfuscation techniques.
- Anti-debug/anti-tamper techniques.
- Investigation of hard-coded secrets or other sensitive information.
- Check for outdated/vulnerable third-party libraries.
- Evaluate weak cryptography implementations.
- Ability to implement biometric authentication bypass.
- Ability to implement certificate pinning bypass.

Specifically not in scope

NowSecure analysts make every attempt to be as thorough as possible in their testing coverage. However, several things are specifically out of scope for our testing:

- Services/software that has no relation to a mobile application.
- Conducting a full web penetration test beyond endpoints exposed to the mobile application.
- Mapping security findings to compliance regulations such as HIPAA or GDPR.
- Developing an exploit for an identified vulnerability.

1.4 Testing Platform

NowSecure performed the assessment using the following devices and OS versions:

Device	Operating System
Pixel 3a	Android 9
iPhone 7	iOS 13.4.1

1.5 Common Vulnerability Scoring System (CVSS)

Every vulnerability listed is assigned a CVSS (Common Vulnerability Scoring System) score. The CVSS score is a community adopted way to quantify risk of a specific vulnerability and apply a weight of severity to that.

Each vulnerability is assigned a risk category of Low, Medium, or High, which is derived from a CVSS (Common Vulnerability Scoring System) score. The following CVSS score ranges and criteria for risk levels are as follows:

LOW (CVSS 0.1 - 3.9)	MEDIUM (CVSS 4.0 - 6.9)	HIGH (CVSS 7.0 - 10.0)
Some information disclosure or potential for exploitation. Considered low risk due to difficulty in discovery/exploitation or limited scope/sensitivity of information.	Significant information disclosure or potential for exploitation. Considered medium risk because a reasonable discovery/exploitation scenario exists, but not high risk.	Sensitive data exposure or security vulnerability exists. Considered high risk because a known attack could harvest sensitive information from one or many users or systems.

1.6 Summary of Findings

Below is a table summarizing all findings resulting from the assessment, ranked from most severe to least. Informational findings are those that don't necessarily constitute a vulnerability, but warrant further investigation or may contain information valuable to the security team receiving this report.

Finding	CVSS Score	Details
2.1 NetworkSecurityConfig Insecure Exception	Medium (5.3)	The Android app allows HTTP usage in its NetworkSecurityConfig file.
2.2 App Transport Security (ATS) Globally Disabled	Medium (5.3)	The iOS app does not use free security protections, AppTransportSecurity. This protection prevents use of weak server ciphers.
2.3 Third-Party Keyboards Enabled	Medium (4)	The iOS app allows use of potentially malicious third party keyboards.
2.4 Use of Default Platform Data Protection	Low (3.9)	The iOS app uses the default security configuration for platform data protections. This free feature enhances data-at-rest security.
2.5 Snapshot Could Reveal Sensitive Information In The Task Switcher	Low (3.3)	The iOS app enables screenshots and UI caching. This can potentially expose sensitive information.
2.6 Auto-Generated Screenshots	Low (3.3)	The Android app enables screenshots and UI caching. This can potentially expose sensitive information.
2.7 Keyboard Cache Potentially Exposing Sensitive Data	Low (2.5)	The iOS app uses functionality that may expose sensitive data to device keyboard cache.
2.8 Sensitive Data Found in Android App Storage	Low (2.3)	The Android app stores sensitive data in clear text database files in its private app storage.
2.9 Stack Smash Protections Disabled	Info	The Android app does not apply Stack Smash Protections to native libraries.
2.10 Cookie Without Secure Flag Set	Info	The app's backend did not set its cookie's secure flag.
2.11 Embedded DEX Feature Flag	Info	The Android app does not use Embedded DEX feature flag, a free security protection.
2.12 Weak Server-Side Ciphers	Info	The app's "notabene" endpoint is configured to allow the use of weak TLS protocols.

1.7 Summary of Recommendations

Avoid Storing Sensitive Information in Device Filesystem / Encrypt All Application Data
Sensitive information should only be stored securely on the device, after encrypting them. Avoid writing any highly sensitive information to the device logs or application files/unencrypted databases. For details on how to protect data in storage, refer to

https://books.nowsecure.com/secure-mobile-development/en/sensitive-data/implement-secure-data-storage.html.

1.8 NowSecure Certification

NowSecure offers a certification to clients for their applications which are assessed and pass the mobile security assessment. The certification enables the app provider to promote user trust and encourage secure application design and verification in the general marketplace. NowSecure provides both a graphical badge and a hosted page providing independent validation of the certification.

With no high risk findings on this assessment, StagingNet-Wallet Staging v0.1/0.5.37 for iOS, Android is eligible for Certification. The app was found through empirical testing to avoid significant security vulnerabilities, and does not store sensitive data insecurely. Certification may be displayed publicly, with a badge and hosted verification page, with agreement to the mobile application security certification terms of use.

2.0 DETAILS OF FINDINGS

2.1 NetworkSecurityConfig Insecure Exception

Category: Network
Applies to: Android
CVSS Score: 5.3
CVSS Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

Finding Description:

The Android app has set an insecure NetworkSecurityConfig exception. The NetworkSecurityConfig allows developers to explicitly allow use of insecure network protocols within android apps. This app was found to allow use of an insecure network protocol.

Finding Details:

The Android app contains NetworkSecurityConfig exceptions that are extraneous to the app, and allow the use of HTTP on the “10.0.2.2” endpoint:

```
<network-security-config xmlns:android="http://schemas.android.com/apk/res/android">
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">10.0.2.2
        </domain>
    </domain-config>
</network-security-config>
```

NetworkSecurityConfig

Attack Scenario:

An attacker may be able to MITM the app by downgrading connections if the app has not been configured properly.

Finding Mitigation:

When creating a NetworkSecurityConfig, avoid using `cleartextTrafficPermitted="true"`, as this can potentially lead to use of HTTP in an app.

In addition, avoiding the use of "user" certs, as this functionality allows untrusted certificates to be used for TLS communications.

References/Resources:

- https://www.nowsecure.com/blog/2018/08/15/a-security-analysts-guide-to-network-security-configuration-in-android-p/

2.2 App Transport Security (ATS) Globally Disabled

Category: Network
CVSS Score: 5.3
Applies to: iOS
CVSS Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

Finding Description:

Your application has globally disabled ATS, which will allow a connection regardless of HTTP or HTTPS configuration, allow connection to servers with lower TLS versions, and allow connection using cipher suites that do not support forward secrecy (FS).

App Transport Security (ATS) is new in iOS 9, and it helps ensure secure connections between an app and the back end server(s). It is on by default when an app is linked against iOS 9.0 SDK or later. With ATS enabled, HTTP connections are forced to use HTTPS (TLS v1.2), and any attempts to connect using insecure HTTP will fail.

Attack Scenario:

An attacker may find more opportunities to attack an app that has not enabled ATS.

Finding Mitigation:

For apps running on iOS 9.0 or higher, ATS must be enabled globally by linking to the iOS 9.0 or later SDK, and avoid setting the "NSAllowsArbitraryLoads" key to "Yes" or "True." For any existing apps which communicate to servers over HTTP, an exception must be set using either the “NSExceptionAllowsInsecureHTTPLoads” or “NSThirdPartyExceptionAllowsInsecureHTTPLoads” key.

References/Resources:

- https://cwe.mitre.org/data/definitions/319.html
- https://forums.developer.apple.com/thread/6767
- https://www.nowsecure.com/blog/2016/12/29/enable-ios-app-transport-security-ats/

2.3 Third-Party Keyboards Enabled

Category: Device
Applies to: iOS
CVSS Score: 4
CVSS Vector: CVSS:3.0/AV:A/AC:H/PR:L/UI:R/S:C/C:L/I:L/A:N

Finding Description:

This application does not use `application:shouldAllowExtensionPointIdentifier:` on the application delegate to disable third-party keyboards. This was determined by testing whether third-party keyboards are allowed at runtime through dynamic analysis.

Attack Scenario:

A third-party keyboard extension that is granted "full access" by the user can send keystrokes to a remote server. A malicious keyboard extension could exfiltrate sensitive data the user enters into the application.

Finding Mitigation:

Implement `application:shouldAllowExtensionPointIdentifier:` on the application delegate and return `NO` for the identifier `UIApplicationKeyboardExtensionPointIdentifier`.

References/Resources:

- https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1623122-application?language=objc
- https://www.owasp.org/index.php/Mobile_Top_10_2016-M1-Improper_Platform_Usage

2.4 Use of Default Platform Data Protection

Category: RE
Applies to: iOS
CVSS Score: 3.9
CVSS Vector: CVSS:3.0/AV:P/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N

Finding Description:

The application was found to implement an improper Data Protection Entitlement based on the data handled by the application. This could expose sensitive information about the device or user to an attacker with access to the device. Apple provides Data Protection Entitlements to protect sensitive data in iOS by encrypting it on disk. Developers can set 4 different levels of protection: 1) No protection (file is always accessible), 2) Complete protection until first user authentication (default), 3) Complete unless already open and 4) Complete (file is accessible only when the device is unlocked). All sensitive user data and device identifiers should be protected at rest on the device.

Attack Scenario:

Sensitive user data on the device may be accessible to an attacker or through forensic analysis.

Finding Mitigation:

Data Protection Entitlements represent the level of data protection that encrypts sensitive user data when accessed on some devices. The level of protection may vary depending on the information that is being handled by the application. Sensitive user data, files containing personal information about the user, or files created directly by the user, always warrant the strongest level of protection (NSFileProtectionComplete), meaning that the file is accessible only when the device is unlocked. Assign the complete protection level to user data files and manage access to those files using the app delegate methods.

References/Resources:

- https://www.owasp.org/index.php/Mobile_Top_10_2016-M1-Improper_Platform_Usage
- https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage
- https://books.nowsecure.com/secure-mobile-development/en/sensitive-data/implement-secure-data-storage.html
- https://pspdfkit.com/blog/2017/how-to-use-ios-data-protection/

2.5 Snapshot Could Reveal Sensitive Information In The Task Switcher

Category: Device
CVSS Score: 3.3

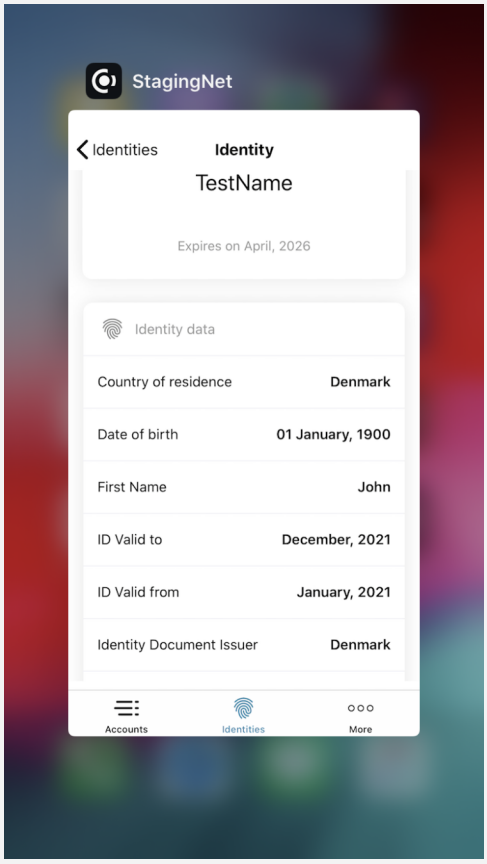
Applies to: iOS
CVSS Vector: CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N

Finding Description:

In order to provide the visual transitions in the interface, iOS has been proven to capture and store snapshots (screenshots or captures) as images stored in the file system portion of the device NAND flash. This occurs when a device suspends (rather than terminates), when either the home button is pressed, or a phone call or other event temporarily suspends the application. These images can often contain user and application data, and in one published case contained the user's full name, DOB, address, employer, and credit scores. For this application, at least one snapshot was stored on the device and could potentially display sensitive information.

Finding Details:

Sensitive data was visible in UI cache:



iOS TaskSwitch with Sensitive Data

Attack Scenario:

An attacker with physical access to the device could see sensitive data on the user's device from apps in the background. An attacker could also retrieve these snapshots from device storage.

Finding Mitigation:

Block caching of application snapshots using API configuration or code in order to protect sensitive data. You can use the willEnterBackground API - When an iOS app is going to be sent to the background, ensure the app is not displaying any sensitive information. Also, create a splash screen for the app and display this as it moves into the background.

References/Resources:

- https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage
- https://developer.apple.com/library/archive/qa/qa1838/_index.html

2.6 Auto-Generated Screenshots

Category: Device
CVSS Score: 3.3

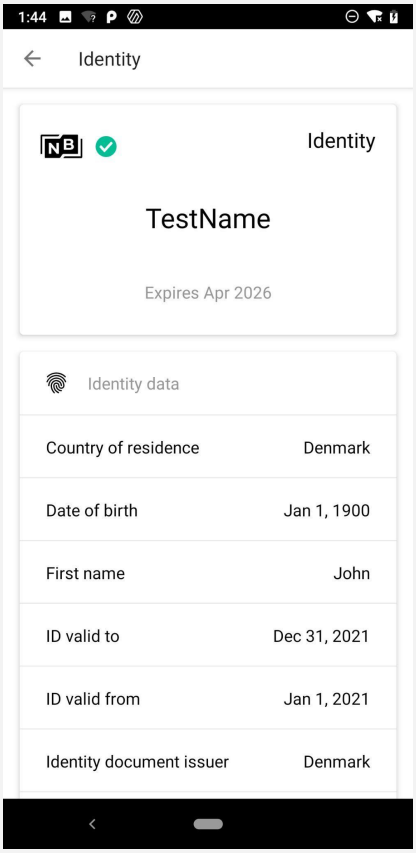
Applies to: Android
CVSS Vector: CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N

Finding Description:

Manufacturers want to provide device users an aesthetically pleasing effect when an application is entered or exited, so the system saves a screenshot when the application goes into the background. This feature could potentially pose a security risk for an application. Sensitive data could be exposed if a user deliberately takes a screenshot of the application while sensitive data is displayed, or in the case of a malicious application running on the device, that is able to continuously capture the screen. This information is written to local storage, from which it may be recovered either by a rogue application on a rooted device, or by someone who steals the device.

Finding Details:

Sensitive data was visible in screenshots and UI cache:



Android app Screenshot

Attack Scenario:

An attacker with physical access to the device could see sensitive data on the user's device from apps in the background. An attacker could also use a malicious app to passively take screenshots during app runtime.

Finding Mitigation:

Remove sensitive information from views before moving to the background. When an application transitions to the background, the system takes a snapshot of the application's main window, which it then presents briefly when transitioning your application back to the foreground. Before returning you should hide or obscure potentially sensitive personal information that might be captured as part of the snapshot. In Android, implement the FLAG_SECURE flag.

References/Resources:

- https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage
- https://developer.apple.com/library/archive/qa/qa1838/_index.html

2.7 Keyboard Cache Potentially Exposing Sensitive Data

Category: Device
Applies to: iOS
CVSS Score: 2.5
CVSS Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:L/I:N/A:N/E:U/RL:O

Finding Description:

During app runtime, an analysis of the app's entry fields is performed. This test flags UI text objects observed with the autocorrectionType set to `UITextAutocorrectionTypeYes` or `UITextAutocorrectionTypeDefault`.

Autocorrection features on iOS may leak potentially sensitive information to other applications or users. Data inserted into text inputs where autocorrection is not explicitly disabled may subsequently be leaked as a consequence.

Data leaked via autocorrect suggestions may be observed by other installed applications, other users with access to the device, or by users shoulder-surfing the user whilst using apps where autosuggestions appear. Depending on the data in question this can impact the user's security and privacy.

Finding Details:

The following fields may expose user data to the UI cache:

```
{
  "class": "UITextField",
  "viewController": "StagingNet.EnterPasswordViewController",
  "autocorrectionType": "UITextAutocorrectionTypeDefault",
  "contentType": "password",
  "secure": "false"
}
```

Insecure UITextField Object

Attack Scenario:

An attacker with physical access can obtain sensitive data from device keyboard cache.

Finding Mitigation:

For text inputs that handle sensitive data, set the 'autocorrectionType' property to `UITextAutocorrectionTypeNo`. This will ensure that data input in such objects is not cached and used as autocorrect suggestions in other apps. Note also that 'UITextAutocorrectionTypeNo' is the default setting for

'secureTextEntry' fields.
For more information, see the UITextAutoCorrectionType documentation.

References/Resources:

- https://github.com/nowsecure/secure-mobile-development/blob/master/en/caching-logging/be-aware-of-the-keyboard-cache.md
- https://developer.apple.com/documentation/uikit/uitextautocorrectiontype?language=objc

2.8 Sensitive Data Found in Android App Storage

Category: Device
Applies to: Android
CVSS Score: 2.3
CVSS Vector: CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:L/I:N/A:N

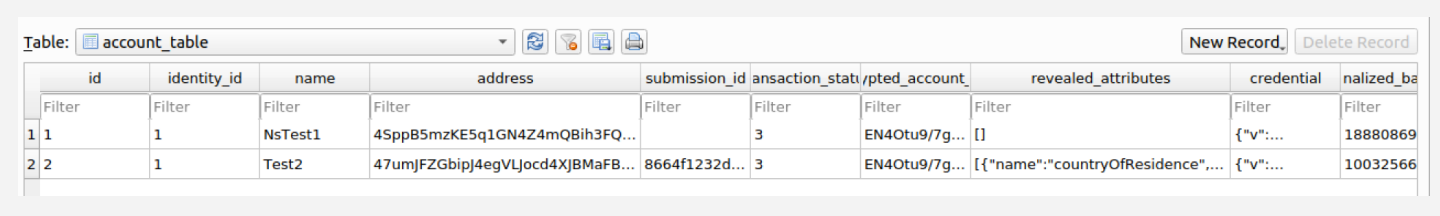
Finding Description:

A forensic acquisition of the test device's file system was obtained. The system was examined for instances of sensitive information including, device ID, serial number, MAC address, email address, and financial or contact data in plaintext, base64, and other common encodings/hashes. Highly sensitive data was found to exist within the app's data directory.

Finding Details:

Sensitive data was discovered in Android private app storage. Sensitive data was discovered in "/com.concordium.wallet.staging/databases/wallet_database" in clear text. Data discovered included the following values:

- Account Name
- Account User
- Address Names and Account
- Account Address
- User First and Last Name



wallet_database Screenshot

Attack Scenario:

An attacker with physical access to the device could recover sensitive data related to the app or the user's sensitive information.

Finding Mitigation:

Sensitive information such as usernames and passwords should not be included in logs, settings files, or unencrypted databases without being protected.

References/Resources:

- https://www.owasp.org/index.php/Mobile_Top_10_2016-M1-Improper_Platform_Usage

- https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage
- https://books.nowsecure.com/secure-mobile-development/en/sensitive-data/implement-secure-data-storage.html

2.9 Stack Smash Protections Disabled

Category: RE **Applies to:** Android

Finding Description:

Stack smashing protection has not been implemented in components included in the application. When an application is compiled with stack smashing protection, a known value or "canary" is placed on the stack directly before the local variables to protect the saved base pointer, saved instruction pointer, and function arguments. The value of the canary is verified upon the function return to see if it has been overwritten. The compiler uses a heuristic to intelligently apply stack protection to a function, typically functions using character arrays. This is a very simple best practice that hardens your app with little to no downside. Memory corruption vulnerabilities can be very hard to track down, but can be extremely severe.

Finding Details:

The following Android app libraries did not apply SSP protections:

```
lib/x86_64/libcurve_arithmetic.so
lib/arm64-v8a/libcrypto_common.so
lib/x86_64/libpedersen_scheme.so
lib/x86_64/libwallet.so
lib/x86_64/libid.so
lib/x86_64/libencrypted_transfers.so
lib/arm64-v8a/libeddsa_ed25519.so
lib/arm64-v8a/libencrypted_transfers.so
lib/arm64-v8a/libbulletproofs.so
lib/x86_64/libeddsa_ed25519.so
lib/arm64-v8a/libps_sig.so
lib/arm64-v8a/libffi_helpers.so
lib/x86_64/libps_sig.so
lib/arm64-v8a/libmobile_wallet.so
lib/x86_64/libmobile_wallet.so
lib/x86_64/librandom_oracle.so
lib/arm64-v8a/libdodis_yampolskiy_prf.so
lib/arm64-v8a/libwallet.so
lib/arm64-v8a/libid.so
lib/arm64-v8a/libelgamal.so
lib/arm64-v8a/libcurve_arithmetic.so
lib/arm64-v8a/librandom_oracle.so
lib/arm64-v8a/libpedersen_scheme.so
lib/x86_64/libelgamal.so
lib/x86_64/libbulletproofs.so
lib/x86_64/libdodis_yampolskiy_prf.so
lib/x86_64/libffi_helpers.so
lib/x86_64/libcrypto_common.so
```

Libs without SSP

Attack Scenario:

This app does not protect against a specific type of attack that can expose the app to an attacker performing custom actions. These custom actions could potentially give them access to sensitive information from the app or the device.

Finding Mitigation:

When creating a NetworkSecurityConfig, avoid using `cleartextTrafficPermitted="true"`, as this can potentially lead to use of HTTP in an app.

Because the Android NDK handles this automatically, it may be worthwhile to switch over to using that capability to manage native libraries. More information can be found at https://developer.android.com/ndk.

If using the provided NDK is not an option, then the issue is likely in the compiler settings for the native libraries that caused the vulnerability to pop up. Make sure that the -fstack-protector-all, -fpic, and -fstack-protector-strong flags are all set in the build.gradle file (typically in the cmake/cppFlags).

References/Resources:

- https://nvd.nist.gov/vuln/detail/CVE-2019-3568
- https://developer.android.com/ndk

2.10 Cookie Without Secure Flag Set

Category: Informational **Applies to:** iOS, Android

Finding Description:

When set to true, the "secure" flag tells the browser to only send the cookie if the request is sent using a secure channel. This will ensure the cookie is not transmitted over unencrypted requests.

Finding Details:

```
Set-Cookie:
_SESSION=QIz4ZFW997rqHCZZuDrE+Qz7aWiDwW7TOkkGNuUesdOnPTgjqWshnCPamdWP/Z7cZwGjudzkDhR+8z/ge
SA8f+eWJN9NjQofbJmk/BoAjkR2NJ51qOAHLec243RtJtB5FQEYZAdcwaUPVtwrjyARNCi0jl5HP2xoNZztBxblW6t
2Zowq/QA=; Path=/; Expires=Wed, 28-Apr-2021 15:34:29 GMT; HttpOnly
```

wallet-proxy.eu.staging.concordium.com Cookie

Attack Scenario:

An attacker might be able to retrieve cookies sent over unencrypted requests.

Finding Mitigation:

It is recommended to enable the "secure" flag to instruct the browser to only send the cookie if the request is sent using a secure channel. There are multiple ways to enable this, one of which is to set it within the response header:

```
Set-Cookie: <name>=<value>; <Max-Age>=<age>; domain=<domain_name>; secure
```

References/Resources:

- https://www.owasp.org/index.php/SecureFlag
- https://www.owasp.org/index.php/Mobile_Top_10_2016-M7-Poor_Code_Quality

2.11 Embedded DEX Feature Flag

Category: Informational **Applies to:** Android

Finding Description:

The embedded DEX feature flag was not enabled in this binary. This security protection available to apps running on Android API level 29 or higher can mitigate attacks where a bad actor is able to tamper locally compiled code on the device by requiring the app to run directly from the APK file.

Finding Mitigation:

This flag can be enabled by setting the android:useEmbeddedDex to true. Please note that this feature may impact the performance of the app, and should be evaluated before enabling it in production.

References/Resources:

- https://developer.android.com/topic/security/dex

2.12 Weak Server-Side Ciphers

Category: Informational **Applies to:** iOS, Android

Finding Description:

The application was found to connect to backend endpoints with weak server side ciphers. Many web servers allow lower encryption settings, such as the very weak, export-grade 40-bit encryption. Implement a strong cipher suite to protect information used in creating shared keys, encrypting messages between clients and servers, and generating message hashes and signatures that ensure the integrity of those messages.

Finding Details:

The "idiss.notabene.studio" endpoint accepts the TLS 1.0 and TLS 1.1 protocol.

Attack Scenario:

An attacker could force an app to use a lesser cipher and compromise that communication. An attacker could also use lesser ciphers to compromise the server backend.

Finding Mitigation:

Ensure SSL certificates are properly installed and configured for the highest encryption possible. If possible, enable only strong ciphers (128-bit and up).

TLSv1 is more than 10 years old and was found vulnerable to a “renegotiation attack” in 2009. Most servers using TLSv1 have been patched to close this vulnerability, but you should verify this for relevant servers. The TLSv1 protocol has been updated and the more current TLSv1.2 offers the latest technology and strongest encryption ciphers available. Updating to the newer version of TLS should harden and future-proof the application.

Avoid weak ciphers, such as:

NULL cipher suite
Anonymous Diffie-Hellmann
DES and RC4 because of their vulnerability to crypto-analytical attacks (NOTE: iOS 10 disables RC4 by default)

Avoid weak protocols, such as:

SSLv2
SSLv3 because of its vulnerability to the POODLE attack - CVE-2014-3566 (NOTE: In iOS 10, the Apple Secure Transport API no longer supports SSLv3)
TLS 1.0 and earlier because the protocols are vulnerable to the CRIME (CVE-2012-4929) and BEAST (CVE-2011-3389) attacks

Reference the OWASP Transport Layer Protection Cheat Sheet for more information about how to securely design and configure transport layer security for an app.

References/Resources:

- https://www.owasp.org/index.php/Testing_for_Weak_SSL/TLS_Ciphers,_Insufficient_Transport_Layer_Protection_(OTG-CRYPST-001)
- https://books.nowsecure.com/secure-mobile-development/en/servers/server-side-ssl-configuration.html
- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.md

3.0 RELEASE INFORMATION

Katie Bochnowski
SVP, Customer Success & Services
kbochnowski@nowsecure.com

Michael Krueger
Director, Application Security
mkrueger@nowsecure.com