



Earth – Wallet

Android & iOS

Mobile App Pentest

Prepared by: Halborn

Date of Engagement: June 12th, 2023 – July 20th, 2023

Visit: Halborn.com

DOCUMENT REVISION HISTORY	9
CONTACTS	9
1 EXECUTIVE OVERVIEW	10
1.1 INTRODUCTION	11
1.2 ASSESSMENT SUMMARY	11
1.3 SCOPE	12
1.4 TEST APPROACH & METHODOLOGY	14
RISK METHODOLOGY	14
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	16
3 FINDINGS & TECH DETAILS ANDROID	18
3.1 (HAL-01) ANDROID - EXPOSURE OF API KEYS THROUGH REVERSE ENGINEERING APPLICATION - HIGH	20
Description	20
Proof of concept	20
CVSS Vector	21
Risk Level	21
Recommendation	21
Remediation Plan	21
3.2 (HAL-02) ANDROID - EXPOSED API KEYS IN APPLICATION REQUESTS - HIGH	22
Description	22
Proof of concept	23
CVSS Vector	25
Risk Level	25
Recommendation	25

Remediation Plan	26
3.3 (HAL-03) ANDROID - DUMP MNEMONICS FROM MEMORY - MEDIUM	27
Description	27
Proof of concept	27
CVSS Vector	28
Risk Level	28
Recommendation	28
Remediation Plan	28
Reference	29
3.4 (HAL-04) ANDROID - CERTIFICATE PINNING BYPASS - MEDIUM	30
Description	30
Proof of concept	31
CVSS Vector	32
Risk Level	32
Recommendation	32
Remediation Plan	32
Reference	32
3.5 (HAL-05) ANDROID - MISCONFIGURATION ALLOWS APPLICATION DATA BACKUP - MEDIUM	33
Description	33
Proof of concept	33
CVSS Vector	34
Risk Level	34
Recommendation	35
Remediation Plan	35
3.6 (HAL-06) ANDROID - CLEARTEXT NETWORK TRAFFIC USAGE IN MOBILE APPLICATION - MEDIUM	36

Description	36
Proof of concept	36
CVSS Vector	37
Risk Level	37
Recommendation	37
Remediation Plan	37
3.7 (HAL-07) ANDROID - LACK OF AUTHENTICATION ON APP STARTUP - LOW	38
Description	38
Proof of concept	38
CVSS Vector	38
Risk Level	38
Recommendation	39
Remediation Plan	39
3.8 (HAL-08) ANDROID- LACK OF ROOT DETECTION MECHANISM - LOW	40
Description	40
CVSS Vector	40
Risk Level	40
Recommendation	40
Remediation Plan	41
References	41
3.9 (HAL-09) ANDROID - LACK OF ANTI-HOOK ANTI-DEBUG MECHANISMS - LOW	42
Description	42
Example Command	42
CVSS Vector	42

	Risk Level	42
	Recommendation	43
	Remediation Plan	43
	References	43
3.10	(HAL-10) ANDROID - FAILURE IN RENDERING TRANSACTION AND BALANCE INFORMATION - LOW	44
	Description	44
	Proof of concept	45
	CVSS Vector	46
	Risk Level	46
	Recommendation	46
	Remediation Plan	47
3.11	(HAL-11) ANDROID - BACKGROUND SCREEN CATCHING - LOW	48
	Description	48
	Proof of concept	49
	CVSS Vector	49
	Risk Level	49
	Recommendation	49
	Remediation Plan	50
4	FINDINGS & TECH DETAILS iOS	51
4.1	(HAL-12) MNEMONICS STORED CLEAR TEXT IN THE KEYCHAIN - HIGH	52
	Description	52
	Proof of concept	52
	CVSS Vector	53
	Risk Level	53
	Recommendation	53

Remediation Plan	53
Reference	53
4.2 (HAL-13) iOS - EXPOSED API KEYS IN APPLICATION REQUESTS - HIGH	54
Description	54
Proof of concept	55
CVSS Vector	57
Risk Level	57
Recommendation	57
Remediation Plan	58
4.3 (HAL-14) iOS - CERTIFICATE PINNING BYPASS - MEDIUM	59
Description	59
Proof of concept	60
CVSS Vector	60
Risk Level	61
Recommendation	61
Remediation Plan	61
Reference	61
4.4 (HAL-15) iOS - SENSITIVE INFORMATION EXPOSURE VIA IOS CLIPBOARD - MEDIUM	62
Description	62
Proof of concept	62
CVSS Vector	63
Risk Level	63

Recommendation	63
Remediation Plan	63
Reference	63
4.5 (HAL-16) iOS - SENSITIVE DATA IN SNAPSHOT - MEDIUM	64
Description	64
Proof of concept	65
CVSS Vector	65
Risk Level	65
Recommendation	65
Remediation Plan	67
4.6 (HAL-17) iOS - DUMP MNEMONICS FROM MEMORY - MEDIUM	68
Description	68
Proof of concept	68
CVSS Vector	68
Risk Level	69
Recommendation	69
Remediation Plan	69
Reference	69
4.7 (HAL-18) iOS - LACK OF AUTHENTICATION ON APP STARTUP - LOW	70
Description	70
Proof of concept	70
CVSS Vector	70
Risk Level	70
Recommendation	71

Remediation Plan	71
4.8 (HAL-19) iOS - LACK OF JAILBREAK DETECTION MECHANISM ON THE iOS APPLICATION - LOW	72
Description	72
CVSS Vector	72
Risk Level	72
Recommendation	72
Remediation Plan	73
Reference	73
4.9 (HAL-20) iOS - LACK OF ANTI-HOOK ANTI-DEBUG MECHANISM ON THE APPLICATION - LOW	74
Description	74
Example Command	74
CVSS Vector	74
Risk Level	75
Recommendation	75
Remediation Plan	75
Reference	75
4.10 (HAL-21) iOS - BACKGROUND SCREEN CATCHING - LOW	76
Description	76
Proof of concept	77
CVSS Vector	77
Risk Level	77
Recommendation	77
Remediation Plan	78
4.11 (HAL-22) iOS - FAILURE IN RENDERING TRANSACTION AND BALANCE INFORMATION - LOW	79
Description	79

	Proof of concept	80
	CVSS Vector	81
	Risk Level	81
	Recommendation	81
	Remediation Plan	82
5	ANNEX	83
5.1	Mobile App Security Testing Methodology	84
	Local Authentication	84
	Data Storage	84
	Network Communication	85
	Cryptographic APIs	86
	Android Description	86
	Anti-Reversing Defenses	86
	Tampering and Reverse Engineering	87
	Input Validation	88
	Server-Side APIs	88

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	07/19/2023
0.2	Draft Review	07/21/2023
0.3	Draft Review	07/21/2023
1.0	Remediation Plan	11/21/2023
1.1	Remediation Plan Review	11/22/2023
1.2	Remediation Plan Review	11/27/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Carlos Polop	Halborn	Carlos.Polop@halborn.com
Afaq Abid	Halborn	Afaq.Abid@halborn.com



EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Earth engaged Halborn to conduct a security assessment on their Android & iOS mobile applications. Beginning on June 12th, 2023 and ending on July 20th, 2023, the security assessment was scoped to Earth Wallet Mobile Apps. The client team provided the source code and their respective APK/IPA files to conduct security testing using tools to scan, detect, validate possible vulnerabilities found in the wallet and report the findings at the end of the engagement.

Though this security assessment's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure mobile app extension development.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided a timeline for the engagement and assigned a full-time security engineer to assess the security of the assets in scope. The security engineer is a penetration testing expert with advanced knowledge in web, mobile, recon, discovery & infrastructure penetration testing.

The goals of our security assessments are to improve the quality of the systems we review and to target sufficient remediation to help protect users.

In summary, Halborn identified some vulnerabilities affecting both Android and iOS versions of the application.

Highly concerning issues include the exposure of API keys through reverse engineering the Android application and the exposure of API keys in application requests, both on Android and iOS. These issues represent a significant risk, with high likelihood and impact levels, due to the potential for unauthorized access to backend services.

A vulnerability has been identified in the way both Android and iOS applications handle mnemonics. Dumping mnemonics from memory or storing them in clear text pose a serious security risk due to the sensitivity of this data.

Other vulnerabilities include the ability to bypass certificate pinning, the misconfiguration that allows application data backup, and the use of cleartext network traffic in the Android application. All these vulnerabilities have the potential to expose sensitive data or weaken the security of the application.

Additional vulnerabilities identified on both Android and iOS include the lack of authentication on app startup, the lack of root or jailbreak detection mechanism, and the lack of anti-hook anti-debug mechanisms. While these vulnerabilities have a lower impact, they increase the application's susceptibility to malicious activities and potentially jeopardize user data.

Issues related to application usability and security have also been identified, including failures in rendering transaction and balance information, and the misuse of background screen catching.

For the iOS application specifically, we noted additional risks associated with sensitive information exposure via the iOS clipboard and sensitive data contained in snapshots.

The Earth team has effectively resolved most of the identified issues, where some identified issues were risk accepted as part of the design and some were partially addressed.

1.3 SCOPE

IN-SCOPE:

The security assessment was scoped to:

- Earth Wallet Mobile Android Package:

- Version [1.2.0](#)
- Earth Wallet Mobile iOS Package:
 - Version [1.2.3](#)
 - Commit/Branch: [66fa6e59314618e18448058e07a3ae2cb8b55dfc](#)

OUT-OF-SCOPE:
External libraries.

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the pentest. While manual testing is recommended to uncover flaws in logic, process and implementation; automated testing techniques assist enhance coverage of the infrastructure and can quickly identify flaws in it.

The following phases and associated tools were used throughout the term of the assessment:

- Storing private keys and assets securely
- Send/Receive tokens and assets securely to another wallet
- Any attack that impacts funds, such as draining or manipulating of funds
- Application Logic Flaws
- Areas where insufficient validation allows for hostile input
- Application of cryptography to protect secrets
- Brute Force Attempts
- Input Handling
- Source Code Review
- Fuzzing of all input parameters
- Technology stack-specific vulnerabilities and Code Assessment
- Known vulnerabilities in 3rd party / OSS dependencies.

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

- 5 - Almost certain an incident will occur.
- 4 - High probability of an incident occurring.
- 3 - Potential of a security incident in the long term.
- 2 - Low probability of an incident occurring.
- 1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

- 5 - May cause devastating and unrecoverable impact or loss.
- 4 - May cause a significant level of impact or loss.
- 3 - May cause a partial impact or loss to many.
- 2 - May cause temporary impact or loss.
- 1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.



- 10 - CRITICAL
- 9 - 8 - HIGH
- 7 - 6 - MEDIUM
- 5 - 4 - LOW
- 3 - 1 - VERY LOW AND INFORMATIONAL

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	4	8	10	0

LIKELIHOOD

(HAL-03) (HAL-17)	(HAL-15) (HAL-16)	(HAL-12)		
(HAL-11) (HAL-21)	(HAL-05) (HAL-06)		(HAL-01) (HAL-02) (HAL-13)	
		(HAL-04) (HAL-14)		
	(HAL-07) (HAL-08) (HAL-09) (HAL-18) (HAL-19) (HAL-20)	(HAL-10) (HAL-22)		

SECURITY ANALYSIS	RISK LEVEL	REMEDATION DATE
(HAL-01) ANDROID - EXPOSURE OF API KEYS THROUGH REVERSE ENGINEERING APPLICATION	High	SOLVED - 09/22/2023
(HAL-02) ANDROID - EXPOSED API KEYS IN APPLICATION REQUESTS	High	SOLVED - 09/22/2023
(HAL-03) ANDROID - DUMP MNEMONICS FROM MEMORY	Medium	SOLVED - 09/22/2023
(HAL-04) ANDROID - CERTIFICATE PINNING BYPASS	Medium	SOLVED - 09/22/2023
(HAL-05) ANDROID - MISCONFIGURATION ALLOWS APPLICATION DATA BACKUP	Medium	SOLVED - 09/22/2023
(HAL-06) ANDROID - CLEARTEXT NETWORK TRAFFIC USAGE IN MOBILE APPLICATION	Medium	SOLVED - 09/22/2023
(HAL-07) ANDROID - LACK OF AUTHENTICATION ON APP STARTUP	Low	RISK ACCEPTED
(HAL-08) ANDROID- LACK OF ROOT DETECTION MECHANISM	Low	SOLVED - 09/22/2023
(HAL-09) ANDROID - LACK OF ANTI-HOOK ANTI-DEBUG MECHANISMS	Low	PARTIALLY SOLVED - 09/22/2023
(HAL-10) ANDROID - FAILURE IN RENDERING TRANSACTION AND BALANCE INFORMATION	Low	SOLVED - 09/22/2023
(HAL-11) ANDROID - BACKGROUND SCREEN CATCHING	Low	SOLVED - 09/22/2023
(HAL-12) MNEMONICS STORED CLEAR TEXT IN THE KEYCHAIN	High	SOLVED - 11/17/2023
(HAL-13) iOS - EXPOSED API KEYS IN APPLICATION REQUESTS	High	SOLVED - 09/28/2023
(HAL-14) iOS - CERTIFICATE PINNING BYPASS	Medium	SOLVED - 09/28/2023
(HAL-15) iOS - SENSITIVE INFORMATION EXPOSURE VIA IOS CLIPBOARD	Medium	SOLVED - 11/21/2023

(HAL-16) iOS - SENSITIVE DATA IN SNAPSHOT	Medium	SOLVED - 09/28/2023
(HAL-17) iOS - DUMP MNEMONICS FROM MEMORY	Medium	SOLVED - 09/28/2023
(HAL-18) iOS - LACK OF AUTHENTICATION ON APP STARTUP	Low	RISK ACCEPTED
(HAL-19) iOS - LACK OF JAILBREAK DETECTION MECHANISM ON THE iOS APPLICATION	Low	SOLVED - 09/28/2023
(HAL-20) iOS - LACK OF ANTI-HOOK ANTI-DEBUG MECHANISM ON THE APPLICATION	Low	PARTIALLY SOLVED - 09/28/2023
(HAL-21) iOS - BACKGROUND SCREEN CATCHING	Low	SOLVED - 09/28/2023
(HAL-22) iOS - FAILURE IN RENDERING TRANSACTION AND BALANCE INFORMATION	Low	SOLVED - 09/28/2023



FINDINGS & TECH DETAILS ANDROID

3.1 (HAL-01) ANDROID - EXPOSURE OF API KEYS THROUGH REVERSE ENGINEERING APPLICATION - HIGH

Description:

The mobile application was found to be containing multiple API keys stored in plaintext within the `BuildConfig` file. These keys were discovered during a reverse engineering process. This exposure of sensitive keys can lead to unauthorized access to the associated services. Leaving API keys embedded in the code, exposing them publicly, can lead to serious vulnerabilities. If an attacker gets access to these keys, they can misuse the permissions granted to these keys, leading to unauthorized access or potentially costly operations if the keys are tied to billing.

Proof of concept:

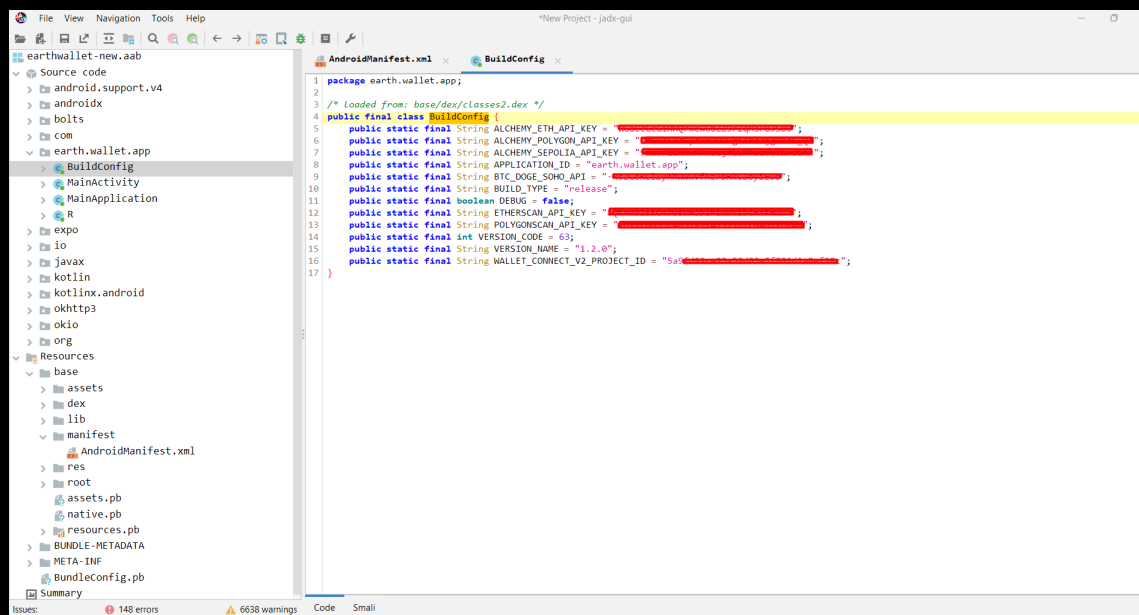


Figure 1: Multiple API keys found during reverse engineering of the application

CVSS Vector:

- **CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N**

Risk Level:**Likelihood - 4****Impact - 4****Recommendation:**

It is recommended to obfuscate your code with tools like proguard, which obfuscate your code by renaming classes, fields, and methods with semantically obscure names, which can make it harder for someone to reverse engineer your app. Along with that, other solution that can be adopted is, avoid storing API keys in your app altogether. One common way to implement this is using a proxy server that your app communicates with, and which in turn communicates with the API endpoints. It is further recommended to revoke all the identified secrets with new one to limit the exposure.

Remediation Plan:

SOLVED: The Earth team solved the issue by removing the identified keys.

NOTE: Retest was conducted on the following version:

- **Version: 1.4.1**

3.2 (HAL-02) ANDROID - EXPOSED API KEYS IN APPLICATION REQUESTS - HIGH

Description:

The mobile application was found to expose services API keys in HTTPS requests when chaining with HAL-03 SSL pinning is bypassed. This exposure can lead to unauthorized access and misuse of the APIs, potentially leading to data leakage, service disruption, or other malicious activities.

SSL Pinning is a security measure used to prevent man-in-the-middle attacks by associating a host with their expected SSL certificate or public key. Bypassing SSL Pinning allows intercepting the application's network traffic, even when it's protected with HTTPS.

In this case, it has been identified that API keys are included in HTTPS requests and can be exposed if SSL Pinning is bypassed. Exposed API keys can lead to unauthorized access to the API, misuse of the application's data and services, and can potentially violate users' privacy or result in financial loss.

Proof of concept:

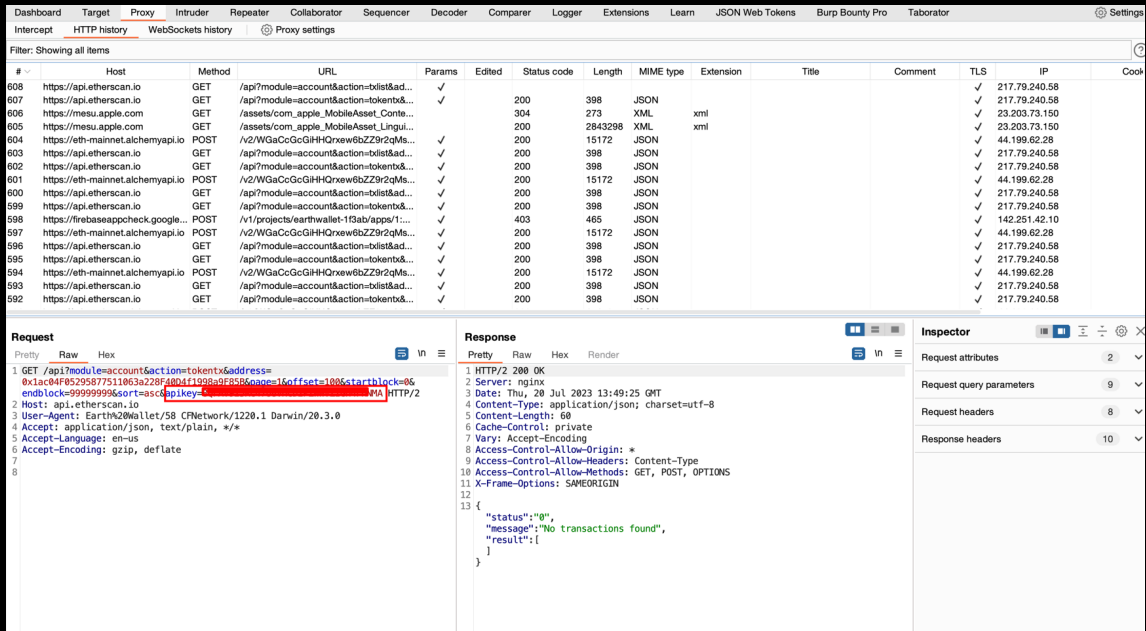


Figure 2: SSL pinning bypassed and application traffic intercepted

3. Intruder attack of https://api.etherscan.io - Temporary attack - Not saved to project file

Results Positions Payloads Resource pool Settings

Filter: Showing all items

Request	Payload	Status code	Error	Timeout	Length	Comment
20	20	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
19	19	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
18	18	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
17	17	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
16	16	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
15	15	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
14	14	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
13	13	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
12	12	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
11	11	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
10	10	200	<input type="checkbox"/>	<input type="checkbox"/>	404	
9	9	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
8	8	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
7	7	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
6	6	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
5	5	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
4	4	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
3	3	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
2	2	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
1	1	200	<input type="checkbox"/>	<input type="checkbox"/>	973	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	973	

Request Response

Pretty Raw Hex

```

1 GET /api?module=account&action=txlist&address=0xe6Fb54342b36E86FDFFD277dCD7Cf37BBaa9215F&startblock=0&sort
  =desc&apikey=[REDACTED] HTTP/2
2 Host: api.etherscan.io
3 Accept: application/json, text/plain, */*
4 Origin: null
5 User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 14_4 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like
  Gecko) Mobile/15E1416
6 Accept-Language: en-us
7 Accept-Encoding: gzip, deflate
8 Connection: close
9
10

```

0 matches

Finished

Figure 3: API keys found in application requests

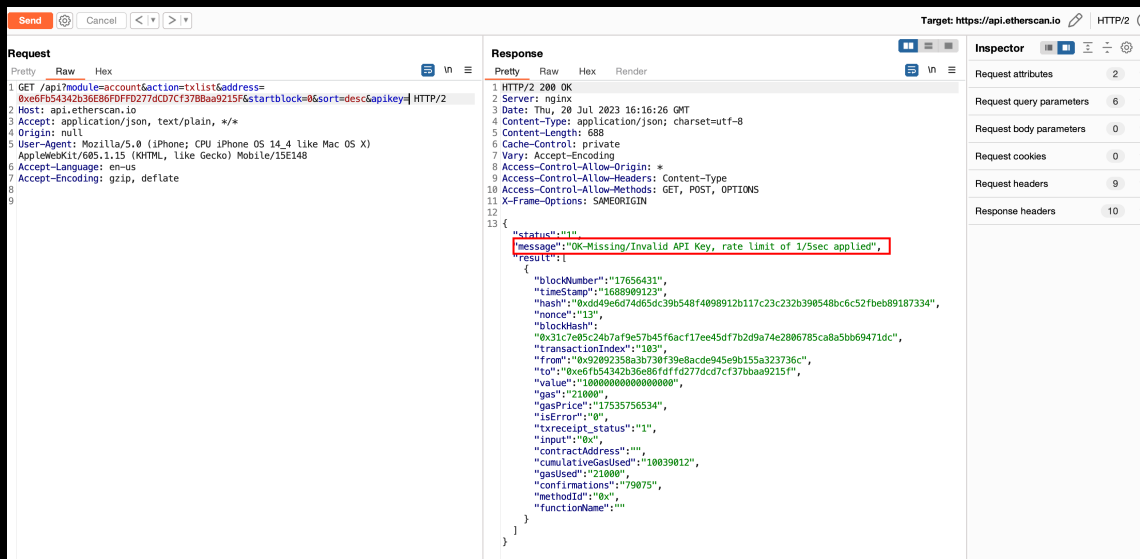


Figure 4: API keys found in application requests

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Risk Level:

Likelihood - 4

Impact - 4

Recommendation:

It is recommended to avoid including services API keys in requests. Instead, use methods like OAuth tokens that are tied to specific users or sessions. Implement server-side controls to limit the actions that can be performed with the API keys. Also recommended reviewing the current implementation of SSL Pinning to prevent bypassing, ensuring the application's network traffic cannot be intercepted easily.

One other common way to implement this is using a proxy server that your app communicates with, and which in turn communicates with the API endpoints. It is further recommended to revoke all the identified secrets with new one to limit the exposure.

Remediation Plan:

SOLVED: The Earth team solved the issue by implementing the appropriate checks.

3.3 (HAL-03) ANDROID - DUMP MNEMONICS FROM MEMORY - MEDIUM

Description:

During the assessment, it was observed that it was possible to dump the mnemonic phrase from the memory of the application and find the mnemonic pattern with regex. As there were no checks against the rooted devices, which makes it possible to dump the running app memory and extract the mnemonics from it.

Note: In the application, `Fridump` was used to dump memory. Our goal was to dump the memory of the application and find the mnemonic pattern with regex.

Proof of concept:

```
PS D:\mobile-audit-tools\frida\fridump3-master> python3 .\fridump3.py -u "Earth Wallet"

Current Directory: D:\mobile-audit-tools\frida\fridump3-master
Output directory is set to: D:\mobile-audit-tools\frida\fridump3-master\dump
Creating directory...
Starting Memory dump...
Oops, memory access violation!-----] 25.41% Complete
Oops, memory access violation!#-----] 40.46% Complete
Oops, memory access violation!#-----] 40.84% Complete
Oops, memory access violation!##-----] 41.15% Complete
Oops, memory access violation!#####-----] 47.05% Complete
Oops, memory access violation!#####-----] 47.18% Complete
Oops, memory access violation!#####-----] 47.24% Complete
Oops, memory access violation!#####-----] 49.18% Complete
Oops, memory access violation!#####-----] 52.01% Complete
Oops, memory access violation!#####-----] 54.58% Complete
Oops, memory access violation!#####-----] 57.4% Complete
Oops, memory access violation!#####-----] 59.91% Complete
Oops, memory access violation!#####-----] 61.48% Complete
Oops, memory access violation!#####-----] 66.62% Complete
Oops, memory access violation!#####-----] 67.06% Complete
Oops, memory access violation!#####-----] 69.01% Complete
Finished! [#####] 99.37% Complete
```

Figure 5: memory dump of Earth wallet application

```

jar verifier does not support more than one entry with the same
consume batched input events but the input event receiver has already been
vertical sync pulse but the display event receiver has already been disposed
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hairX
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair
march humble galaxy chapter vehicle spice great grace cream shove juice hair

```

Figure 6: mnemonics in memory

CVSS Vector:

- CVSS:3.1/AV:P/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:N

Risk Level:

Likelihood - 1

Impact - 5

Recommendation:

It is recommended to have protected measures against jailbreak detection and some open-source tools like Frida in the application to prevent loading/running the application if these tools are detected on the device.

Remediation Plan:

SOLVED: The Earth team has resolved the issue by integrating jailbreak/root detection into the application's build.

Reference:

- [OWASP Tampering and Reverse Engineering](#)
- [OWASP Root Detection Methods](#)
- [OWASP Android Lack of binary protections](#)

3.4 (HAL-04) ANDROID - CERTIFICATE PINNING BYPASS - MEDIUM

Description:

Certificate pinning is the process of associating the backend server with a particular X.509 certificate or public key, instead of accepting any certificate signed by a trusted certificate authority (CA). After storing (“pinning”) the server’s certificate or public key, the mobile app will subsequently connect only to the known server. Withdrawing trust from external CAs reduces the attack surface (after all, there are many cases of CAs being compromised or tricked into issuing certificates to impostors).

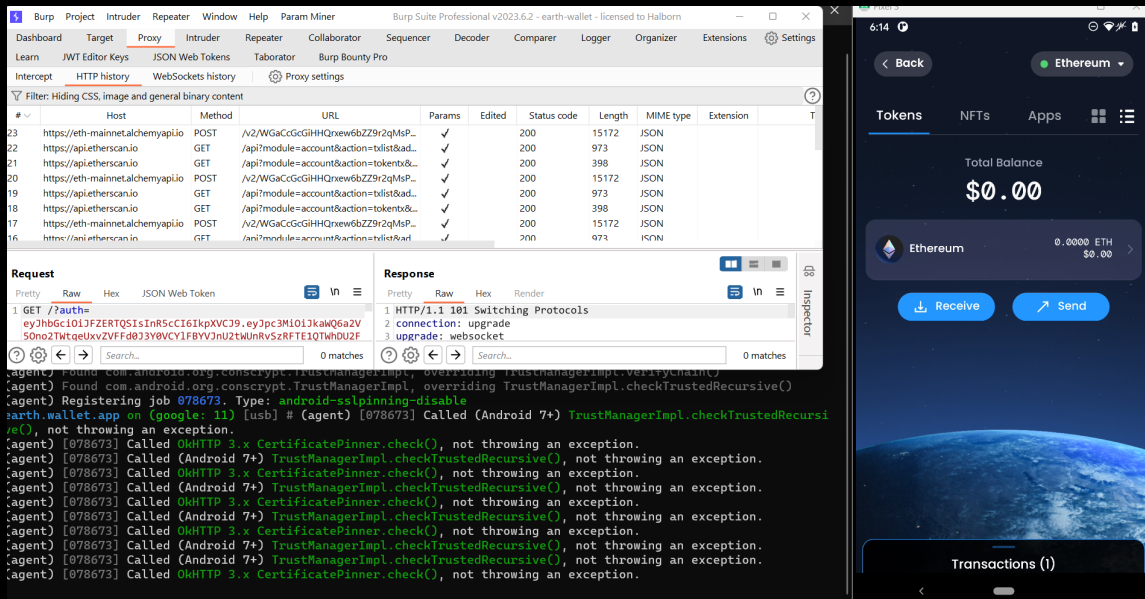
The certificate can be pinned and hardcoded in the app or retrieved at the time the app first connects to the backend. In the latter case, the certificate is associated (“pinned” to) the host when the host is first seen. This alternative is less secure because attackers intercepting the initial connection can inject their certificates.

The target application has not correctly implemented SSL pinning when establishing a trusted connection between the mobile applications and the back-end web services. Without enforcing SSL pinning, an attacker could man-in-the-middle the connection between mobile applications and back-end web services. This allows an attacker to sniff user credentials, session ID, etc. Certificate pinning is used in modern applications to prevent users from intercepting and analyzing HTTP traffic. Using this method, an application can verify the server’s certificate and, in case there is a Man-in-The-Middle, not trust any other certificate than the one stored as default. There are many ways to perform this security countermeasure, and taking it in place does not ensure that a motivated attacker will be able to bypass it in time, but it does represent the first wall of defense against HTTP attacks.

However, in the case of **Earth Wallet** android, although it implements SSL pinning, it uses methods with common names and does not implement anti-hooking mechanisms, which allows attackers to bypass this protection

and make it possible to steal the authentication token used in requests as well.

Proof of concept:



1. Connect to the application using Frida and Objection

```
Listing 1
1 objection --gadget <package-name> explore
```

2. Set the automatic certificate pinning bypass implemented by objection

```
Listing 2
1 android sslpinning disable
```

As it can be seen above, the `certificatePinner.check()` method of `OkHTTP` and `CheckTrustedRecursive()` of `TrustManagerImpl` are triggered and modified at runtime:

CVSS Vector:

- [CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N](#)

Risk Level:**Likelihood - 3****Impact - 3****Recommendation:**

It is recommended to prevent these actions by enforcing anti-tampering and anti-debugging mechanisms. This vulnerability is related to rooting detection and anti-debug and anti-tampering (following). Having methods that cannot be triggered by name and anti-hooking, debugging and rooting detection mechanisms should be enough to start preventing certificate pinning bypass. Additionally, an application should follow the following best practices:

- Set an HTTP Public Key Pinning (HPKP) policy that is communicated to the client application and/or supports HPKP in the client application, if applicable.

Remediation Plan:

SOLVED: The Earth team addressed the issue by implementing the jailbreak/root detection in the build of application.

Reference:

- [Android Certificate Pinning](#)
- [OWASP Pinning Cheat Sheet](#)
- [Android Code Obfuscation](#)
- [Guidelines Towards Secure SSL Pinning in Mobile Applications](#)

3.5 (HAL-05) ANDROID - MISCONFIGURATION ALLOWS APPLICATION DATA BACKUP - MEDIUM

Description:

The mobile application is found to be allowing data backups, as indicated by the `android:allowBackup="true"` setting in the `AndroidManifest.xml` file. This can lead to potentially sensitive information being exposed if the backup data is mishandled or accessed by malicious parties.

The application's data backup setting is configured to permit data backups. This can lead to the unintended exposure of sensitive data if the backup is not securely stored or if it is shared across different contexts that have varying security levels.

Proof of concept:

The screenshot shows an IDE window with the `AndroidManifest.xml` file open. The `android:allowBackup="true"` attribute is visible in the `android:android:allowBackup` tag on line 59. The code is as follows:

```

1 | <?xml version="1.0" encoding="utf-8"?>
2 | <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="earth.wallet.app" android:versionCode="63" android:versionName="1.0" android:allowBackup="true" android:label="@string/app_name" android:icon="@mipmap/ic_launcher" android:theme="@style/AppTheme">
3 |     <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="31"/>
4 |     <uses-permission android:name="android.permission.INTERNET"/>
5 |     <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
6 |     <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
7 |     <uses-permission android:name="android.permission.VIBRATE"/>
8 |     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
9 |     <uses-permission android:name="android.permission.CAMERA"/>
10 |     <queries>
11 |         <intent>
12 |             <action android:name="android.intent.action.VIEW"/>
13 |             <category android:name="android.intent.category.BROWSABLE"/>
14 |             <data android:scheme="https"/>
15 |         </intent>
16 |         <intent>
17 |             <action android:name="android.intent.action.OPEN_DOCUMENT_TREE"/>
18 |         </intent>
19 |     </queries>
20 |     <uses-permission android:name="android.permission.WAKE_LOCK"/>
21 |     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
22 |     <uses-feature android:name="android.hardware.camera" android:required="false"/>
23 |     <uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
24 |     <uses-permission android:name="android.permission.USE_BIOMETRIC"/>
25 |     <uses-permission android:name="android.permission.USE_FINGERPRINT"/>
26 |     <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
27 |     <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"/>
28 |     <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
29 |     <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
30 |     <uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
31 |     <application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:icon="@mipmap/ic_launcher">
32 |         <meta-data android:name="expo.modules.updates.ENABLED" android:value="true"/>
33 |         <meta-data android:name="expo.modules.updates.EXPO_UPDATE_URL" android:value="https://u.expo.dev/f093e318-7afe-48ec-b534-9c2c1c1c1c1c"/>
34 |         <meta-data android:name="expo.modules.updates.EXPO_RUNTIME_VERSION" android:value="1.0.0"/>
35 |         <activity android:configChanges="keyboard|keyboardHidden|orientation|screenSize|uiMode" android:exported="true" android:label="@string/app_name" android:launchMode="singleTask">
36 |             <intent-filter>
37 |                 <action android:name="android.intent.action.MAIN"/>
38 |                 <category android:name="android.intent.category.LAUNCHER"/>
39 |             </intent-filter>
40 |             <intent-filter>
41 |                 <action android:name="android.intent.action.VIEW"/>

```

Figure 7: Application allows backups

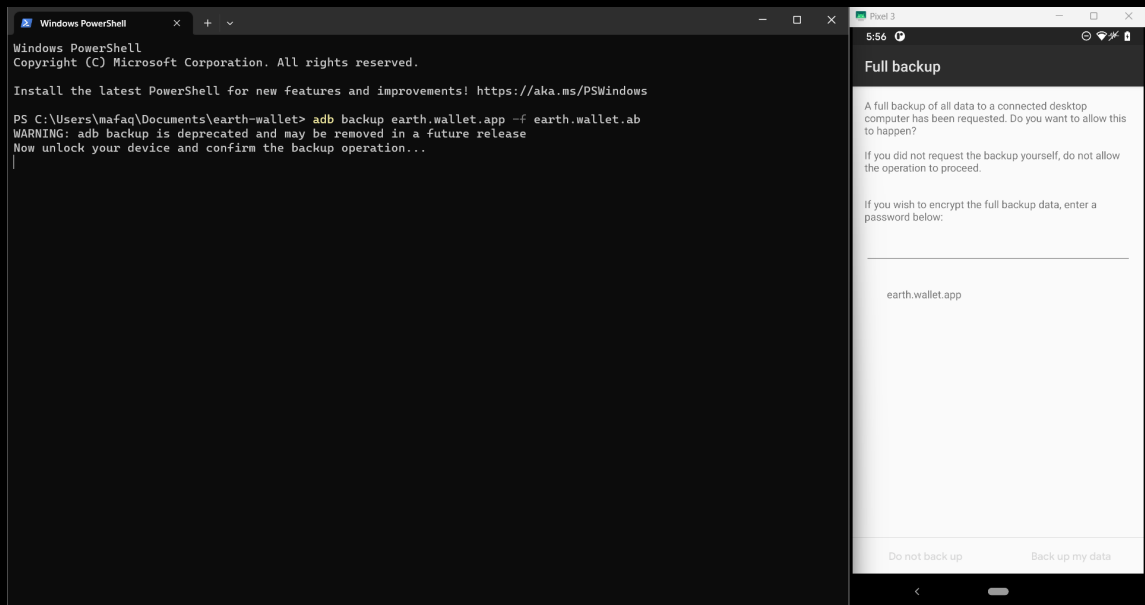


Figure 8: Backing up the application data

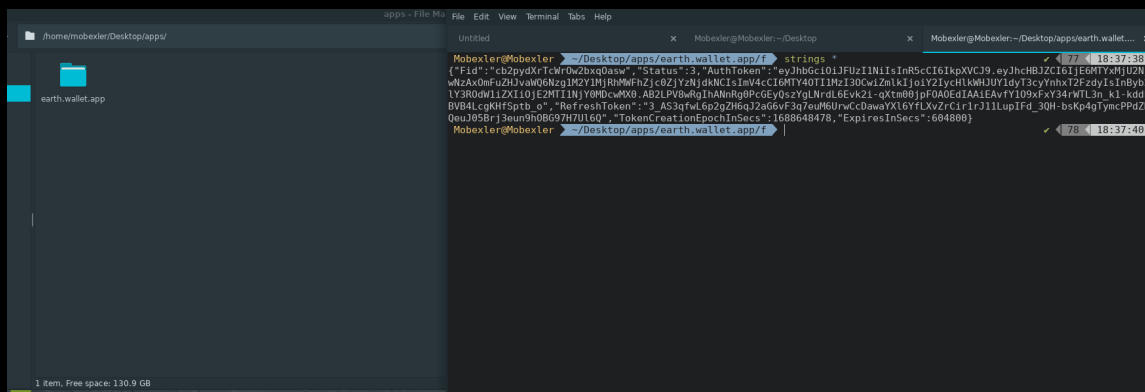


Figure 9: Found an authenticated token in the application backup

CVSS Vector:

- CVSS:3.1/AV:P/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N

Risk Level:

Likelihood - 2

Impact - 4

Recommendation:

It is recommended to set `android:allowBackup="false"` in the `AndroidManifest.xml` to remediate it.

Remediation Plan:

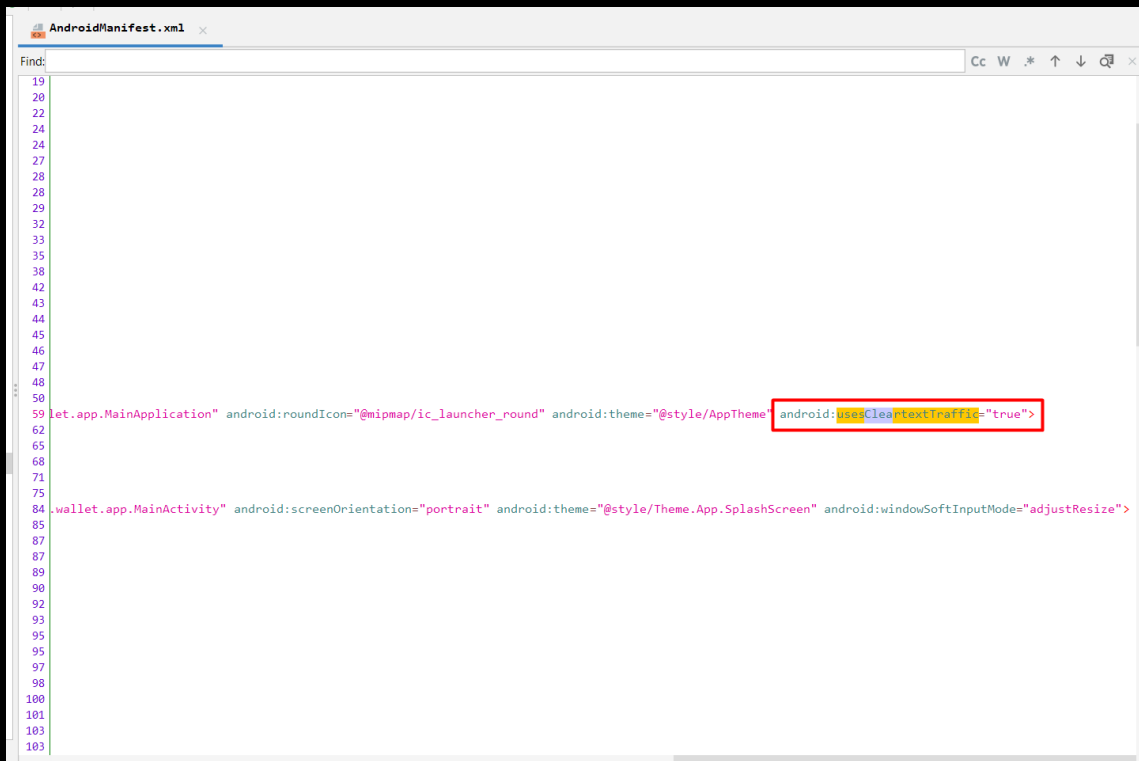
SOLVED: The Earth team solved the issue as per the recommendation.

3.6 (HAL-06) ANDROID - CLEARTEXT NETWORK TRAFFIC USAGE IN MOBILE APPLICATION - MEDIUM

Description:

During the assessment, it has been discovered that the mobile application can communicate over cleartext network traffic (HTTP), as identified by the `android:usesCleartextTraffic="true"` setting in the `AndroidManifest.xml` file. This value indicates that the application intends to use cleartext network traffic, such as cleartext HTTP, without the Android system throwing any errors or warnings. This is generally not recommended because it can expose your app to risks associated with insecure network traffic, such as data interception.

Proof of concept:



```
19
20
22
24
24
27
28
28
29
32
33
35
38
42
43
44
45
46
47
48
50
59 <activity android:name=".app.MainActivity" android:roundIcon="@mipmap/ic_launcher_round" android:theme="@style/AppTheme" android:usesCleartextTraffic="true">
62
65
68
71
75
84 <activity android:name=".wallet.app.MainActivity" android:screenOrientation="portrait" android:theme="@style/Theme.App.SplashScreen" android:windowSoftInputMode="adjustResize">
85
87
87
89
90
92
93
95
95
97
98
100
101
103
103
```

Figure 10: Android application allowed to communicate over insecure network

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 2

Impact - 4

Recommendation:

It is recommended to set `android:usesCleartextTraffic="false"` in `AndroidManifest.xml` and ensure that all network traffic is encrypted using HTTPS or other secure protocols.

Remediation Plan:

SOLVED: The Earth team solved the issue as per the recommendation.

3.7 (HAL-07) ANDROID - LACK OF AUTHENTICATION ON APP STARTUP - LOW

Description:

The mobile application was found to lack biometric or password authentication upon startup, despite having these security measures in place for money transactions and viewing the seed phrase. This potentially exposes sensitive user information to unauthorized users who have physical access to the device.

The application currently does not prompt for any form of biometric or password authentication upon launch. This means that an individual who gains physical access to a device where the app is installed can view the account balance and other potentially sensitive information without any additional security barriers.

While it is commendable that the application requests biometric or password authentication for transactions and reviewing the seed phrase, it is a security best practice to also require this level of authentication at app startup to prevent unauthorized access to sensitive information.

Proof of concept:

Loom Video: [Lack of authentication on application startup](#)

CVSS Vector:

- CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

It is recommended to introduce biometric or password authentication on the application startup. This step will secure potentially sensitive data from unauthorized users who gain physical access to the device. Additionally, consider integrating an automatic timeout to log users out after periods of inactivity, further minimizing risks.

Remediation Plan:

RISK ACCEPTED: The Earth team acknowledged and accepted the risk associated with the current finding, aligning with their design principles.

3.8 (HAL-08) ANDROID- LACK OF ROOT DETECTION MECHANISM - LOW

Description:

Anti-root mechanisms are not used in the Android applications. These mechanisms can help mitigate reverse engineering, application modification, and unauthorized versions of mobile applications to some extent, but few if any will be completely successful against a determined adversary. However, they can be used as part of a defense-in-depth strategy that seeks to minimize the impact and likelihood of such an attack, along with binary patching, local resource modification, method hooking, method swizzling, and heap modification.

CVSS Vector:

- CVSS:3.1/AV:P/AC:H/PR:H/UI:R/S:C/C:L/I:L/A:N

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

The application should detect rooting methods to prevent modifications to the app. As a security best practice, it is recommended to implement a mechanism to check the rooted status of the mobile device. This can be done either manually by implementing a custom solution or using libraries already built for this purpose. This can be done by searching for commonly known files and locations, checking file permissions and attempting to find common rooting services like SuperSU, Magisk or OpenSSH, for example.

Remediation Plan:

SOLVED: The Earth team has resolved the issue by integrating jailbreak/root detection into the application's build.

References:

- [OWASP Tampering and Reverse Engineering](#)
- [OWASP Root Detection Methods](#)
- [OWASP Android Lack of binary protections](#)
- [Android SafetyNet Attestation API](#)

3.9 (HAL-09) ANDROID - LACK OF ANTI-HOOK ANTI-DEBUG MECHANISMS - LOW

Description:

The tested application does not have any security features or mechanisms to prevent malicious actions, Anti Hook and Anti Debug mechanisms.

Example Command:

- Install Frida on the rooted phone. [Frida for Android](#)
- Use the Objection Tool to investigate the Anti-Hook mechanisms in the application. [Objection](#)
- Use the following command in the objection tool to investigate the rooted device.

Listing 3

```
1 objection -g <package-name> explore --startup-command "android
↳ hooking watch class_method <package-name>.MainActivity.onCreate"
```

You can see that the application does not terminate; therefore the application does not have anti-hook or anti-tampering mechanisms.

CVSS Vector:

- [CVSS:3.1/AV:P/AC:H/PR:H/UI:R/S:C/C:L/I:L/A:N](#)

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

Anti-Debug, Anti-Hook and Integrity Check mechanism (completed in the native code), will protect against injection of various types of scripts into it, i.e., Frida Gadgets. The application should not allow modifications in its operation.

Remediation Plan:

PARTIALLY SOLVED: The Earth team partially addressed the issue by implementing the jailbreak/root detection in the build of application.

References:

- [OWASP Reverse Engineering and Tampering](#)
- [AppKnox anti debugging techniques](#)

3.10 (HAL-10) ANDROID - FAILURE IN RENDERING TRANSACTION AND BALANCE INFORMATION - LOW

Description:

The mobile application was found to be experiencing a functional issue where transaction information and Ethereum balance are not correctly rendered in the UI. This issue compromises the usability of the application and could potentially cause confusion or misinterpretation of information among users.

The application currently does not correctly render transaction data and balance information for Ethereum. This can lead to a user being unable to verify their transaction history or accurately assess their account balance, both crucial aspects of cryptocurrency management.

This issue appears to be rooted in the application's UI rendering process rather than a security flaw. Nevertheless, it's essential to address this problem promptly, as it impacts the overall user experience and could lead to misunderstandings or errors in financial management.

Proof of concept:

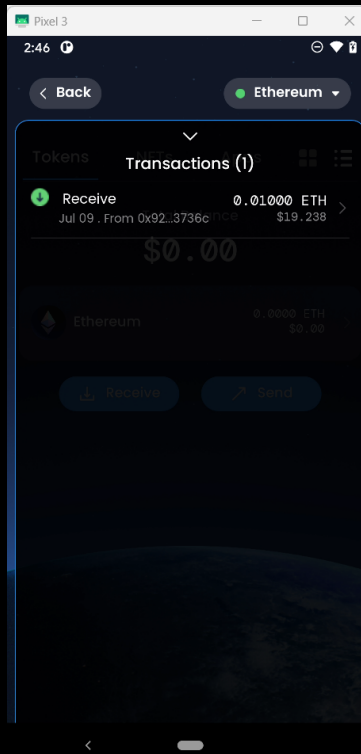


Figure 11: Transaction in application history

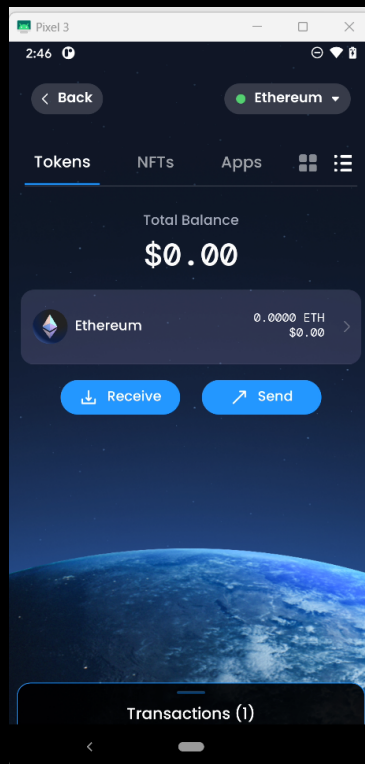


Figure 12: Failed to render balance amount on UI

CVSS Vector:

- CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

Risk Level:

Likelihood - 3

Impact - 2

Recommendation:

It is recommended to investigate the root cause of the rendering issue and implement appropriate fixes to ensure correct display of balance and transaction data. Conduct thorough testing to ensure the issue is fully resolved and the application displays accurate data consistently.

Remediation Plan:

SOLVED: The Earth team fixed the issue by implementing the appropriate checks.

3.11 (HAL-11) ANDROID – BACKGROUND SCREEN CATCHING – LOW

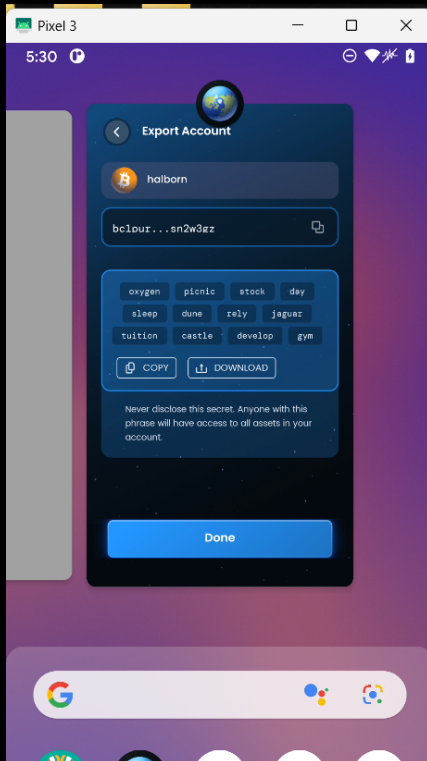
Description:

Manufacturers want to provide device users with an aesthetically pleasing experience at application startup and exit, so they introduced the screenshot-saving feature for use when the application has been executed in the background. This feature may pose a security risk since a user can deliberately screenshot the application while sensitive data is being displayed. A malicious application that is running on the device and able to continuously capture the screen may also expose data. Screenshots are written to local storage, from which they may be recovered by a rogue application (if the device is rooted) or someone who has stolen the device.

For example, capturing a screenshot of a banking application may reveal information about the user's account, credit, transactions, and so on.

In this specific case of the android application, the mnemonic phrase can be captured while the application is not active.

Proof of concept:



CVSS Vector:

- CVSS:3.1/AV:P/AC:L/PR:H/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 1

Impact - 4

Recommendation:

This vulnerability represents a minimal exposure to exploitation. Only the users of the mobile devices to which the attacker has access are affected by this vulnerability.

As a best practice, consider preventing run background screen caching if the application displays sensitive data.

Remediation Plan:

SOLVED: The Earth team fixed the issue by implementing the appropriate checks.



FINDINGS & TECH DETAILS IOS

4.1 (HAL-12) MNEMONICS STORED CLEAR TEXT IN THE KEYCHAIN - HIGH

Description:

The iOS mobile keychain is a secure storage mechanism designed to store sensitive data, such as passwords, tokens, and private keys. However, if an application stores the mnemonic phrase and private keys in clear text format in the iOS mobile keychain, it could potentially expose this sensitive information to attackers if they gain access to the device.

Storing sensitive data in clear text format in the iOS mobile keychain is a serious security vulnerability, as it allows attackers to easily access and steal the mnemonic phrase and private keys without any additional authentication or encryption. This could lead to theft of funds or unauthorized access to user accounts.

Proof of concept:

1. Create an account through an iOS application.
2. Use the following code to explore an application via the objection tool.

Listing 4

```
1 objection -g <package.name> explore
```

3. Type the following command for dumping keychain.

Listing 5

```
1 ios keychain dump
```

4. In the keychain, password stored clearly.

```

4d908e80872432e0f2402a077282feda1b9efcf232aff8eff531e8ebdb6bfb0f0a2eb78225ede82e6e733d83f05ea0e8b9840bab7fe6192a53c0cb45cb5095fca9077ebf679d613ab7d054ae16a4f2280e5abec9cf3b
547916b6ffbb26e3e99bd88e80a6ddeb717b87d5dffca91cf3e53227e88df83bd4b41c41e93f54d99d8d93defcdadf750175c094979c31fa77acc0a28970d4d8f7e9c5a7fed6f3d001820f1404cd4384cbc7f6356bf
ad4baca70378071b92c57a15e904b2f94573800980f6aa5e19640ddca21c8c7b3d5f1ed391c87987a9eca2dd4a04961b34efc21eac931a1
2023-06-13 12:55:08 +0000 WhenPasscodeSetThisDeviceOnly kSecAccessControlBiometryCurrentSet Password bc1q1795wz8t5pavsc3zcqjcv0shw0jt6ymwStj5 bc1q1795w
cv0shw0jt6ymwStj5 smart address smoke very feed what eternal fluid defense horror issue civil
2022-06-07 18:49:39 +0000 AfterFirstUnlock None Password signingCertificatePassword com.riley
da1ec04a-8c9a-4b29-ae17-5a73230b9b32
2022-06-07 18:49:05 +0000 AfterFirstUnlock None Password patreonCreatorAccessToken com.riley

```

Figure 13: Mnemonic found stored in cleartext format in Keychain

CVSS Vector:

- [CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H](#)

Risk Level:

Likelihood - 3

Impact - 5

Recommendation:

To address this vulnerability, the application should ensure that the mnemonic phrase and private keys and related sensitive information are not stored in clear text format in the iOS mobile keychain. Instead, the sensitive information should be encrypted using a secure encryption algorithm and stored in the keychain.

- Make sure object references are set correctly in the Key chain.
- The flags on the keychain should be reviewed.
- Jailbreak detection should be implemented in the application.

Remediation Plan:

SOLVED: The Earth team fixed the issue by implementing the appropriate checks.

Reference:

- [Restricting Keychain Item Accessibility](#)
- [iOS Application Security](#)

4.2 (HAL-13) iOS - EXPOSED API KEYS IN APPLICATION REQUESTS - HIGH

Description:

The mobile application was found to expose services API keys in HTTPS requests when chaining with HAL-12 SSL pinning is bypassed. This exposure can lead to unauthorized access and misuse of the APIs, potentially leading to data leakage, service disruption, or other malicious activities.

SSL Pinning is a security measure used to prevent man-in-the-middle attacks by associating a host with their expected SSL certificate or public key. Bypassing SSL Pinning allows intercepting the application's network traffic, even when it's protected with HTTPS.

In this case, it has been identified that API keys are included in HTTPS requests and can be exposed if SSL Pinning is bypassed. Exposed API keys can lead to unauthorized access to the API, misuse of the application's data and services, and can potentially violate users' privacy or result in financial loss.

Proof of concept:

The image displays a proof of concept for bypassing SSL pinning on an iOS application. It is divided into three main sections:

- Terminal Output:** Shows the execution of a Burp Suite agent on an iPhone. The logs indicate that the agent successfully bypassed SSL pinning by hooking various framework methods (NSURLSession, TLS, BoringSSL) and registering a custom callback for `SSL_CTX_set_custom_verify()`. The output includes:


```
Agent injected and responds ok!
[tab] for command suggestions
earth.wallet.mobile.54HUYYWG37 on (iPhone: 14.4) [usb] # ios sslpinning disable
(agent) Hooking common framework methods.
(agent) Found NSURLSession based classes. Hooking known pinning methods.
(agent) Hooking lower level SSL methods.
(agent) Hooking lower level TLS methods.
(agent) Hooking BoringSSL methods.
(agent) Registering job 225496. Type: ios-sslpinning-disable
earth.wallet.mobile.54HUYYWG37 on (iPhone: 14.4) [usb] # (agent) [225496] Called tls_helper_create_peer_trust(), returning noErr
(agent) [225496] Called SSL_CTX_set_custom_verify(), setting custom callback.
earth.wallet.mobile.54HUYYWG37 on (iPhone: 14.4) [usb] # (agent) [225496] Called SSL_CTX_set_custom_verify(), setting custom callback.
```
- Burp Suite Interface:** Shows the 'HTTP history' tab with a table of intercepted requests. The selected request is a POST to `https://firebaseappcheck.google.../v1/projects/earthwallet-1f3ab/apps/1...`. The 'Inspector' tab shows the response, which is an `HTTP/2 500 Internal Server Error`.

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension
82	https://api.etherscan.io	GET	/api?module=account&action=blst&ad...		✓	200	398	JSON	
81	https://api.etherscan.io	GET	/api?module=account&action=tokenrft...		✓	200	398	JSON	
80	https://firebaseappcheck.google...	POST	/v1/projects/earthwallet-1f3ab/apps/1...		✓	403	465	JSON	LTXM
- Ethereum Mobile App:** Shows the app's interface with a 'Total Balance' of `$0.00` and '0.0000 ETH' displayed. The app has navigation buttons for 'Receive' and 'Send'.

Figure 14: SSL pinning bypassed and application traffic intercepted

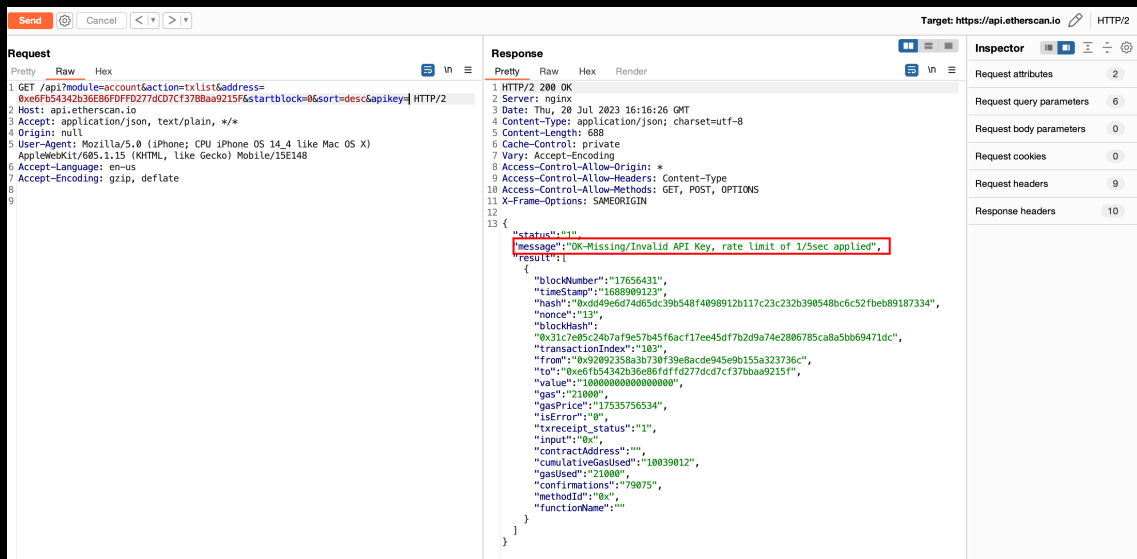


Figure 16: API keys found in application requests

CVSS Vector:

- `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N`

Risk Level:

Likelihood - 4

Impact - 4

Recommendation:

It is recommended to avoid including services API keys in requests. Instead, use methods like OAuth tokens that are tied to specific users or sessions. Implement server-side controls to limit the actions that can be performed with the API keys. Also recommended reviewing the current implementation of SSL Pinning to prevent bypassing, ensuring the application's network traffic cannot be intercepted easily.

One other common way to implement this is using a proxy server that your app communicates with, and which in turn communicates with the API endpoints. It is further recommended to revoke all the identified secrets with new one to limit the exposure.

Remediation Plan:

SOLVED: The Earth team fixed the issue by implementing the appropriate checks.

4.3 (HAL-14) iOS - CERTIFICATE PINNING BYPASS - MEDIUM

Description:

Certificate pinning is the process of associating the backend server with a particular X.509 certificate or public key, instead of accepting any certificate signed by a trusted certificate authority (CA). After storing (“pinning”) the server’s certificate or public key, the mobile app will subsequently connect only to the known server. Withdrawing trust from external CAs reduces the attack surface (after all, there are many cases of CAs being compromised or tricked into issuing certificates to impostors).

The certificate can be pinned and hardcoded in the app or retrieved at the time the app first connects to the backend. In the latter case, the certificate is associated (“pinned” to) the host when the host is first seen. This alternative is less secure because attackers intercepting the initial connection can inject their certificates.

The target application has not correctly implemented SSL pinning when establishing a trusted connection between the mobile applications and the back-end web services. Without enforcing SSL pinning, an attacker could man-in-the-middle the connection between mobile applications and back-end web services. This allows an attacker to sniff user credentials, session ID, etc. Certificate pinning is used in modern applications to prevent users from intercepting and analyzing HTTP traffic. Using this method, an application can verify the server’s certificate and, in case there is a Man-in-The-Middle, not trust any other certificate than the one stored as default. There are many ways to perform this security countermeasure, and taking it in place does not ensure that a motivated attacker will be able to bypass it in time, but it does represent the first wall of defense against HTTP attacks.

However, in the case of **Earth Wallet**, although it implements SSL pinning, it uses methods with common names and does not implement anti-hooking mechanisms, which allows attackers to bypass this protection and make it

possible to steal the authentication token used in requests as well.

Proof of concept:

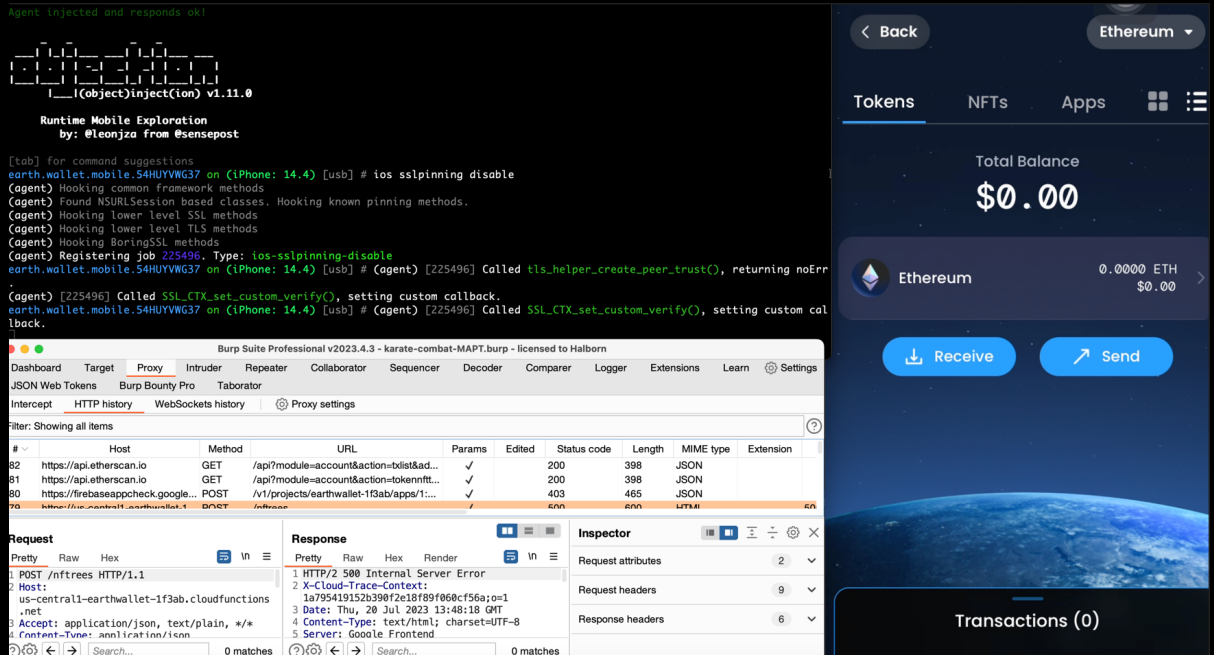


Figure 17: SSL Pinning Bypass

1. Connect to the application using Frida and Objection

Listing 6

```
1 objection --gadget "<package-name>" explore
```

2. Set the automatic certificate pinning bypass implemented by objection

Listing 7

```
1 ios sslpinning disable
```

CVSS Vector:

- CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:**Likelihood - 3****Impact - 3****Recommendation:**

The application should detect jailbreak to prevent these types of attacks. As a security best practice, it is recommended to implement a mechanism to check the rooted status of the mobile device. This can be done either manually by implementing a custom solution or using libraries already built for this purpose. This can be done by searching for commonly known files and locations, checking file permissions and attempting to find common rooting services like Cydia, su, OpenSSH etc.

Remediation Plan:

SOLVED: The Earth team has resolved the issue by integrating jailbreak/root detection into the application's build.

Reference:

- [OWASP Pinning Cheat Sheet](#)
- [OWASP Jailbreak Detection Methods](#)
- [IOSSecuritySuite](#)

4.4 (HAL-15) iOS - SENSITIVE INFORMATION EXPOSURE VIA IOS CLIPBOARD - MEDIUM

Description:

The iOS application allows users to copy the seed phrase to the iOS clipboard. This can introduce potential security risks, as other malicious apps could potentially access and compromise sensitive information stored in the clipboard. Additionally, if the user has iCloud clipboard enabled, the seed phrase could be accessible on other devices connected to the same iCloud account.

If a malicious actor gains access to the user's seed phrase, they could potentially gain access to the user's cryptocurrency wallets or other accounts that rely on that seed phrase for authentication. This could result in the theft of funds or other sensitive data, as well as damage to the user's reputation and trust in your application.

Proof of concept:

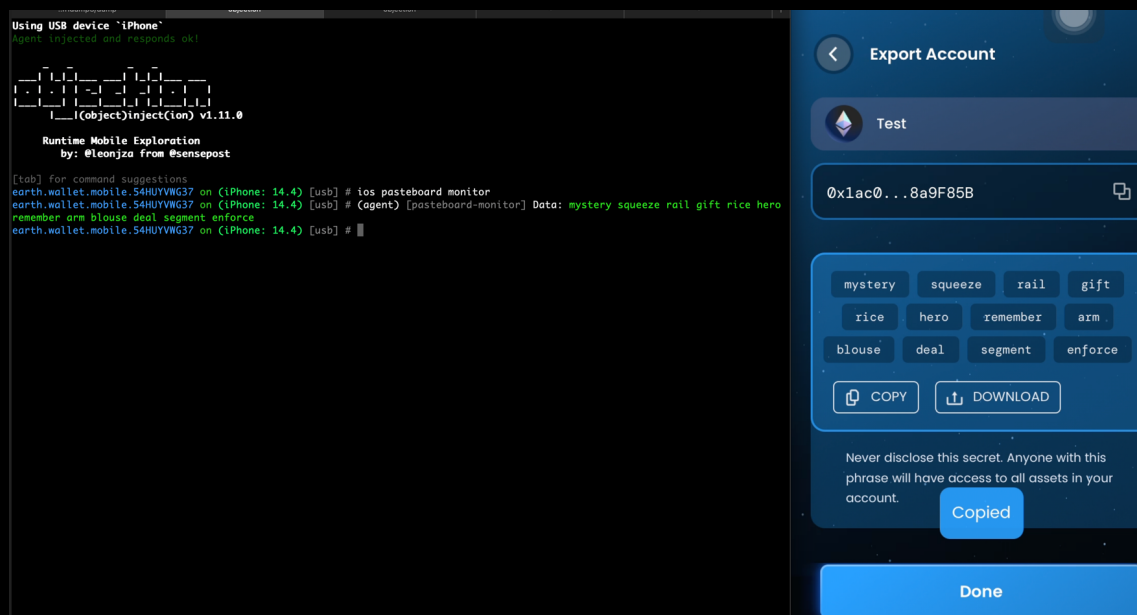


Figure 18: Mnemonic phrase captured from iOS Clipboard

CVSS Vector:

- [CVSS:3.1/AV:N/AC:H/PR:H/UI:R/S:U/C:H/I:H/A:N](#)

Risk Level:**Likelihood - 2****Impact - 5****Recommendation:**

To mitigate this vulnerability, it is recommended to consider disabling the ability to copy the seed phrase to the clipboard. Instead, alternative methods can be implemented for users to securely store their seed phrase, such as:

- Implementing a clipboard timeout for sensitive data, such as seed phrases, will automatically clear the information from the clipboard after a specified period of time.
- Allowing users to export the seed phrase as an encrypted file, which can then be stored on an external storage device or a secure cloud storage service.
- Integrating your app with hardware wallets or other secure storage solutions to store the seed phrase.
- Encouraging users to write down the seed phrase on a piece of paper and store it in a secure location.

Remediation Plan:

SOLVED: The Earth team fixed the issue by implementing the timeout on clipboard to prevent the exposure.

Reference:

- [OWASP Sensitive Data Exposure](#)

4.5 (HAL-16) iOS - SENSITIVE DATA IN SNAPSHOT - MEDIUM

Description:

During the analysis, it has been observed that sensitive data like seed-phrase and private-key can be saved as a snapshot in iOS. Whenever you press the home button, iOS takes a snapshot of the current screen to be able to do the transition to the application in a much smoother way. However, if sensitive data is present in the current screen, it will be saved in the image (which persists across reboots). These are the snapshots that you can also access, by double tapping the home screen to switch between apps.

Unless the iPhone is jailbroken, the attacker needs to have access to the device unblocked to see these screenshots. By default, the last snapshot is stored in the application's sandbox in `Library/Caches/Snapshots/` or `Library/SplashBoard/Snapshots` folder and can leak the sensitive details related to the user's wallet.

Proof of concept:

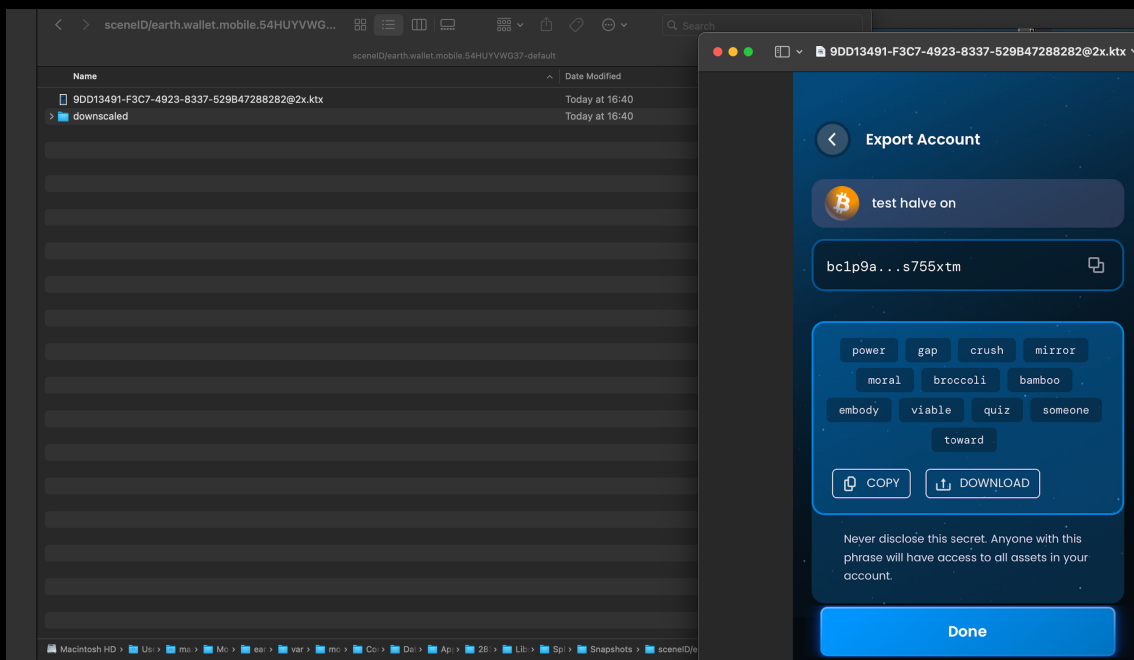


Figure 19: Snapshot in iOS contains sensitive details

CVSS Vector:

- CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Risk Level:

Likelihood - 2

Impact - 5

Recommendation:

It is recommended to prevent sensitive data leaks in snapshots. One way to prevent this bad behavior is to put a blank screen or remove the sensitive data before taking the snapshot using the `ApplicationDidEnterBackground()` function. The following is a sample remediation method that will set a default screenshot.

Swift:

Listing 8

```

1 private var backgroundImage: UIImageView?
2
3 func applicationDidEnterBackground(_ application: UIApplication) {
4     let myBanner = UIImageView(image: #imageLiteral(resourceName:
↳ "overlayImage"))
5     myBanner.frame = UIScreen.main.bounds
6     backgroundImage = myBanner
7     window?.addSubview(myBanner)
8 }
9
10 func applicationWillEnterForeground(_ application: UIApplication)
↳ {
11     backgroundImage?.removeFromSuperview()
12 }

```

Objective-C:

Listing 9

```

1 @property (UIImageView *)backgroundImage;
2
3 - (void)applicationDidEnterBackground:(UIApplication *)application
↳ {
4     UIImageView *myBanner = [[UIImageView alloc] initWithImage:@"
↳ overlayImage.png"];
5     self.backgroundImage = myBanner;
6     self.backgroundImage.bounds = UIScreen.mainScreen.bounds;
7     [self.window addSubview:myBanner];
8 }
9
10 - (void)applicationWillEnterForeground:(UIApplication *)
↳ application {
11     [self.backgroundImage removeFromSuperview];
12 }

```

This sets the background image to overlayImage.png whenever the application is back grounded. It prevents sensitive data leaks because overlayImage.png will always override the current view.

Remediation Plan:

SOLVED: The Earth team fixed the issue by implementing the appropriate checks.

4.6 (HAL-17) iOS - DUMP MNEMONICS FROM MEMORY - MEDIUM

Description:

During the assessment, it was observed that it was possible to dump the mnemonic phrase from the memory of the application and find the mnemonic pattern with regex. As there were no checks against the jailbroken devices, which makes it possible to dump the running app memory and extract the mnemonics from it.

Note: In the application, `Fridump` was used to dump memory. Our goal was to dump the memory of the application and find the mnemonic pattern with regex.

Proof of concept:

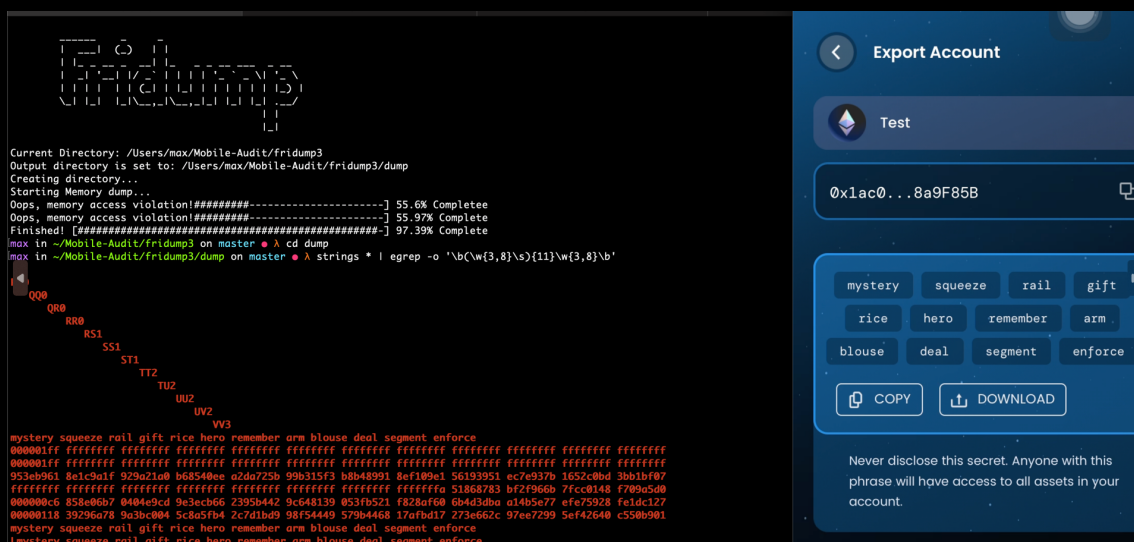


Figure 20: mnemonics found in memory dump

CVSS Vector:

- CVSS:3.1/AV:P/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:N

Risk Level:

Likelihood - 1

Impact - 5

Recommendation:

It is recommended to have protected measures against jailbreak detection and some open-source tools like Frida in the application to prevent loading/running the application if these tools are detected on the device.

Remediation Plan:

SOLVED: The Earth team has resolved the issue by integrating jailbreak/root detection into the application's build.

Reference:

- [iOS Tampering and Reverse Engineering](#)
- [OWASP Jailbreak Detection Methods](#)
- [IOSSecuritySuite](#)

4.7 (HAL-18) iOS - LACK OF AUTHENTICATION ON APP STARTUP - LOW

Description:

The mobile application was found to lack biometric or password authentication upon startup, despite having these security measures in place for money transactions and viewing the seed phrase. This potentially exposes sensitive user information to unauthorized users who have physical access to the device.

The application currently does not prompt for any form of biometric or password authentication upon launch. This means that an individual who gains physical access to a device where the app is installed can view the account balance and other potentially sensitive information without any additional security barriers.

While it is commendable that the application requests biometric or password authentication for transactions and reviewing the seed phrase, it is a security best practice to also require this level of authentication at app startup to prevent unauthorized access to sensitive information.

Proof of concept:

Loom Video: [Lack of authentication on application startup](#)

CVSS Vector:

- CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

It is recommended to introduce biometric or password authentication on the application startup. This step will secure potentially sensitive data from unauthorized users who gain physical access to the device. Additionally, consider integrating an automatic timeout to log users out after periods of inactivity, further minimizing risks.

Remediation Plan:

RISK ACCEPTED: The Earth team has acknowledged and accepted the risk associated with the current finding, aligning with their design principles.

4.8 (HAL-19) iOS - LACK OF JAILBREAK DETECTION MECHANISM ON THE iOS APPLICATION - LOW

Description:

Anti-jailbreak mechanisms are not used in the iOS application. These mechanisms can help mitigate reverse engineering, application modification, and unauthorized versions of mobile applications to some extent, but few if any will be completely successful against a determined adversary. However, they can be used as part of a defense-in-depth strategy that seeks to minimize the impact and likelihood of such an attack, along with binary patching, local resource modification, method hooking, method swizzling, and heap modification.

CVSS Vector:

- [CVSS:3.1/AV:P/AC:H/PR:H/UI:R/S:C/C:L/I:L/A:N](#)

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

The application should not allow any modifications in its operation.

- Obfuscated code.
- Add Frida and other open-source jailbreak detection tools.
- Implement jailbreak detection.

Remediation Plan:

SOLVED: The Earth team resolved the issue by integrating jailbreak/root detection into the application's build.

Reference:

- [iOS Tampering and Reverse Engineering](#)
- [objection - Runtime Mobile Exploration](#)
- [iOS Platform Security & Anti-tampering Swift Library](#)

4.9 (HAL-20) iOS - LACK OF ANTI-HOOK ANTI-DEBUG MECHANISM ON THE APPLICATION - LOW

Description:

The tested application does not have any security features or mechanisms to prevent malicious actions, Anti Hook and Anti Debug.

Example Command:

- Install Frida on the jailbroken phone. [Setup Jailbroken Device](#)
- Use the Objection Tool to investigate the Anti-Hook mechanisms in the application. [Objection](#)
- Use the following command in the objection tool to investigate the Jailbroken device.

Listing 10

```
1 objection --gadget "<package name>" explore
```

- Run the following code in the objection.

Listing 11

```
1 # ios nsuserdefaults get
```

- You can see an application does not terminate; therefore, the application does not have anti-hook or anti-tampering mechanisms.

CVSS Vector:

- [CVSS:3.1/AV:P/AC:H/PR:H/UI:R/S:U/C:L/I:L/A:N](#)

Risk Level:**Likelihood - 2****Impact - 2****Recommendation:**

Anti-Debug, Anti-Hook and Integrity Check mechanism (completed in the native code), which will protect against injection of various types of scripts into it, i.e., Frida Gadgets. The application should not allow modifications in its operation.

Remediation Plan:

PARTIALLY SOLVED: The Earth team partially addressed the issue by implementing the jailbreak detection in the build of application.

Reference:

- [Owasp MSTG](#)
- [IOS Security Suite](#)

4.10 (HAL-21) iOS - BACKGROUND SCREEN CATCHING - LOW

Description:

Manufacturers want to provide device users with an aesthetically pleasing experience at application startup and exit, so they introduced the screenshot-saving feature for use when the application has been executed in the background. This feature may pose a security risk since a user can deliberately screenshot the application while sensitive data is being displayed. A malicious application that is running on the device and able to continuously capture the screen may also expose data. Screenshots are written to local storage, from which they may be recovered by a rogue application (if the device is rooted) or someone who has stolen the device.

For example, capturing a screenshot of a banking application may reveal information about the user's account, credit, transactions, and so on.

In this specific, the mnemonic phrase can be captured while the application is not active.

Proof of concept:

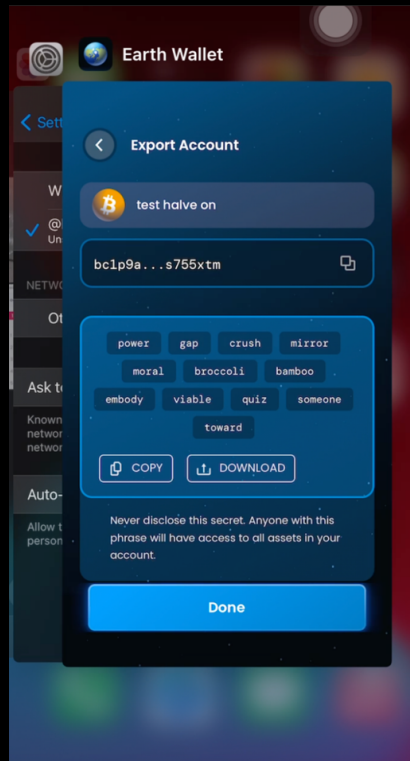


Figure 21: Background screen cache

CVSS Vector:

- `CVSS:3.1/AV:P/AC:L/PR:H/UI:N/S:U/C:L/I:N/A:N`

Risk Level:

Likelihood - 1

Impact - 4

Recommendation:

This vulnerability represents a minimal exposure to exploitation. Only the users of the mobile devices to which the attacker has access are affected by this vulnerability.

As a best practice, consider preventing run background screen caching if

the application displays sensitive data.

Remediation Plan:

SOLVED: The Earth team fixed the issue by implementing the appropriate checks.

4.11 (HAL-22) iOS - FAILURE IN RENDERING TRANSACTION AND BALANCE INFORMATION - LOW

Description:

The mobile application was found to be experiencing a functional issue where transaction information and Ethereum balance are not correctly rendered in the UI. This issue compromises the usability of the application and could potentially cause confusion or misinterpretation of information among users.

The application currently does not correctly render transaction data and balance information for Ethereum. This can lead to a user being unable to verify their transaction history or accurately assess their account balance, both crucial aspects of cryptocurrency management.

This issue appears to be rooted in the application's UI rendering process rather than a security flaw. Nevertheless, it's essential to address this problem promptly, as it impacts the overall user experience and could lead to misunderstandings or errors in financial management.

Proof of concept:

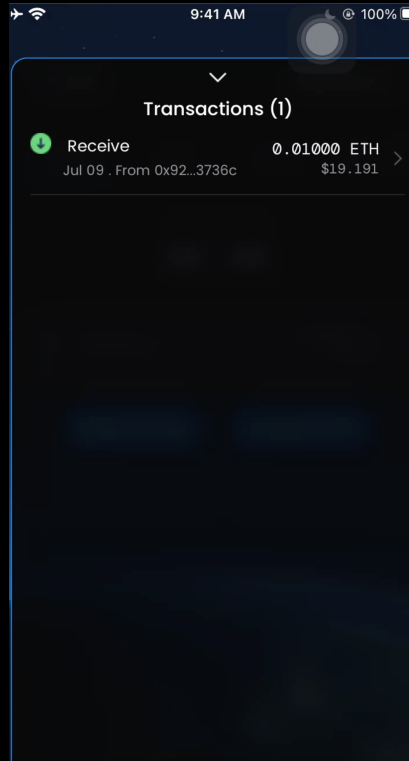


Figure 22: Transaction in application history

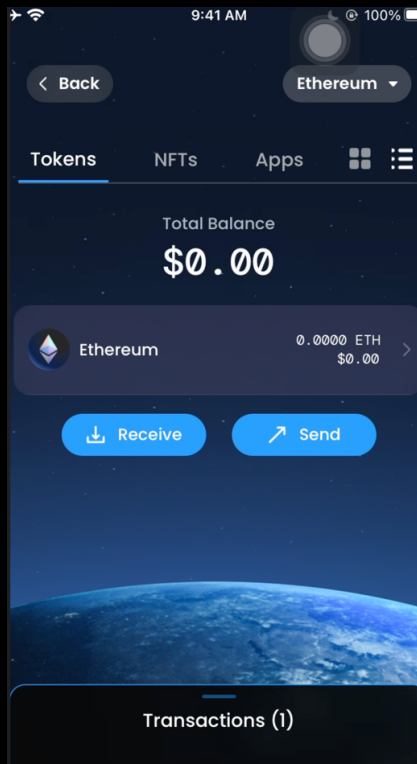


Figure 23: Failed to render balance amount on UI

CVSS Vector:

- CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

Risk Level:

Likelihood - 3

Impact - 2

Recommendation:

It is recommended to investigate the root cause of the rendering issue and implement appropriate fixes to ensure correct display of balance and transaction data. Conduct thorough testing to ensure the issue is fully resolved and the application displays accurate data consistently.

Remediation Plan:

SOLVED: The Earth team fixed the issue by implementing the appropriate checks.



ANNEX



5.1 Mobile App Security Testing Methodology

Local Authentication:

During local authentication, an app authenticates the user against credentials stored locally on the device. In other words, the user “unlocks” the app or some inner layer of functionality by providing a valid PIN, password or biometric characteristics such as face or fingerprint, which is verified by referencing local data. Generally, this is done so that users can more conveniently resume an existing session with a remote service or as a means of step-up authentication to protect some critical function.

On Android, there are two mechanisms supported by the Android Runtime for local authentication: the Confirm Credential flow and the Biometric Authentication flow.

Fingerprint authentication on iOS is known as Touch ID. The fingerprint ID sensor is operated by the SecureEnclave security coprocessor and does not expose fingerprint data to any other parts of the system. Next to Touch ID, Apple introduced Face ID: which allows authentication based on facial recognition. Both use similar APIs on an application level, the actual method of storing the data and retrieving the data (e.g. facial data or fingerprint related data is different).

The tests performed include enforced password/PIN strength requirements, 2FA, and re-authentication to ensure that the application correctly enforced password/PIN strength requirements, 2FA is functioning as expected and re-authentication is correctly being prompted during sensitive operations.

Data Storage:

Android Description

Protecting authentication tokens, private information, and other

sensitive data is key to mobile security. In this chapter, you will learn about the APIs Android offers for local data storage and best practices for using them.

The guidelines for saving data can be summarized easily: public data should be available to everyone, but sensitive and private data must be protected, or, better yet, kept out of device storage.

iOS Description

The protection of sensitive data, such as authentication tokens and private information, is key for mobile security. In this chapter, you'll learn about the iOS APIs for local data storage, and best practices for using them.

As little sensitive data as possible should be saved in permanent local storage. However, in most practical scenarios, at least some user data must be stored. Fortunately, iOS offers secure storage APIs, which allow developers to use the cryptographic hardware available on every iOS device. If these APIs are used correctly, sensitive data and files can be secured via hardware-backed 256-bit AES encryption.

The tests performed include storage of application data, data encryption, Keychain/Keystore access control, clipboard, sensitive data stored, snapshots, and many more to ensure that all stored passwords/PINs are properly encrypted, private keys are securely stored in an encrypted format, the proper access control is implemented to limit the accessibility of stored sensitive data, the clipboard is automatically cleared, etc.

Network Communication:

Almost every Android/iOS app acts as a client to one or more remote services. As this network communication usually takes place over untrusted networks such as public Wi-Fi, classical network based-attacks become a potential issue.

Most modern mobile apps use variants of HTTP-based web services, as these protocols are well-documented and supported.

The tests performed include traffic inspection, HTTPS communications, and cache to ensure that the application should use HTTPS or another secure protocols, the application does not store sensitive information in logs, cache, or analytics data that could potentially be accessed by other applications or attackers.

Cryptographic APIs:

iOS Description

Apple provides libraries that include implementations of the most common cryptographic algorithms. Apple's Cryptographic Services Guide is a great reference. It contains generalized documentation of how to use standard libraries to initialize and use cryptographic primitives, information that is useful for source code analysis.

Android Description:

Android cryptography APIs are based on the Java Cryptography Architecture (JCA). JCA separates the interfaces and implementation, making it possible to include several security providers that can implement sets of cryptographic algorithms. Most of the JCA interfaces and classes are defined in the `java.security.*` and `javax.crypto.*` packages. In addition, there are Android-specific packages `android.security.*` and `android.security.keystore.*`.

KeyStore and KeyChain provide APIs for storing and using keys (behind the scene, KeyChain API uses KeyStore system). The tests performed included

The tests performed include cryptographic methods' usage, and memory analysis to ensure the application uses a secure and up-to-date algorithm, and key material is not leaked in memory and is securely wiped from memory after use.

Anti-Reversing Defenses:

Description

The lack of these measures does not cause a vulnerability - instead, they

are meant to increase the app's resilience against reverse engineering and specific client-side attacks.

The tests performed include jailbroken/rooted devices' detection, reverse engineer tasks on the code, binary manipulation, and debug scenario to ensure that the application detects a jailbroken iOS device or a rooted Android device and responds accordingly, obfuscation techniques are employed to make reverse-engineering more difficult, mechanism to detect binary tampering or modification are implemented, etc.

Tampering and Reverse Engineering:

iOS Description

iOS reverse engineering is a mixed bag. On one hand, apps programmed in Objective-C and Swift can be disassembled nicely. In Objective-C, object methods are called via dynamic function pointers called "selectors", which are resolved by name during runtime. The advantage of runtime name resolution is that these names need to stay intact in the final binary, making the disassembly more readable. Unfortunately, this also means that no direct cross-references between methods are available in the disassembler and constructing a flow graph is challenging.

Android Description

Android's openness makes it a favorable environment for reverse engineers. In the following chapter, we'll look at some peculiarities of Android reversing and OS-specific tools as processes.

Android offers reverse engineers big advantages that are not available with iOS. Because Android is open-source, you can study its source code at the Android Open-Source Project (AOSP) and modify the OS and its standard tools any way you want. Even on standard retail devices, it is possible to do things like activating developer mode and sideloading apps without jumping through many hoops. From the powerful tools shipping with the SDK to the wide range of available reverse engineering tools, there are a lot of niceties to make your life easier.

The tests performed include injecting snippets with Frida to test the application's defense, assessment errors to examine the application's

response to ensure that the application detects these tools and responded as expected, and error messages should not disclose sensitive information or information that could aid an attacker.

Input Validation:

For any publicly accessible data storage, any process can override the data. This means that input validation needs to be applied the moment the data is read back again.

The tests performed include different input injections and how the application handles the malicious inputs to ensure that the application sanitizes and validates all user inputs before processing to prevent potential attacks, mitigating the risk of injection attacks.

Server-Side APIs:

Attacks targeting APIs are one of the most serious security threats facing businesses, as they provide direct access to sensitive data and functionalities. And attackers have become aware of the popularity of APIs and the existence of critical vulnerabilities in these interfaces. The problem is that web applications remain the primary target of attacks and APIs now represent 90% of the attack surface of web applications. Thus, APIs have become one of the main attack vectors, with devastating financial consequences for the companies that bear the costs.

The tests performed include server-side API testing, and high-load tests in the backend to ensure that the API is not vulnerable to vulnerabilities such as business logic vulnerabilities, access control, authentication, etc. and the application should handle unexpected volumes of data or requests gracefully, without crashing or becoming unresponsive and not observing any application crashes or slowdowns.



THANK YOU FOR CHOOSING

// HALBORN

