

PUBLIC ACCESS

CYBERSECURITY AUDIT REPORT

Version v1.1

This document details the process and results of the penetration test performed by CyStack on behalf of Rice from 11/02/2022 to 01/03/2022.

Prepared for

Rice Technologies Limited Liability Company

Prepared by

Vietnam CyStack Joint Stock Company

© 2022 CyStack. All rights reserved.

Portions of this document and the templates used in its production are the property of CyStack and cannot be copied (in full or in part) without CyStack's permission.

While precautions have been taken in the preparation of this document, CyStack the publisher, and the author(s) assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein. Use of CyStack's services does not guarantee the security of a system, or that computer intrusions will not occur.

Contents

1 Executive Summary	4
1.1 Key Findings	4
1.2 Limitations	4
1.3 Assessment Components	5
2 Dashboard	6
3 Code Review Details	8
3.1 Steps to Conduct	8
3.2 Results	9
4 Recommendations	14
5 Asset Metadata	15
6 Vulnerability Details	22
7 Appendix	27
Appendix A – Vulnerability Severity Ratings	27
Appendix B – Vulnerability Categories	28
Appendix C – Security Assessment Based On OWASP Web Security Testing Guide v4.2	29
Appendix D – Security Assessment Based On OWASP Mobile Security Testing Guide v1.2	34
Mobile Application Security Requirements	34
Resiliency against Reverse Engineering	40

Confidentiality Statement

This document is the exclusive property of **Rice Technologies Limited Liability Company (Rice)** and **CyStack Vietnam Joint Stock Company (CyStack)**. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both **Rice** and **CyStack**.

CyStack may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. CyStack prioritized the assessment to identify the weakest security controls an attacker would exploit. CyStack recommends Rice conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

Version History

Version	Date	Release notes
1.0	01/03/2022	Report with open issues
1.1	16/03/2022	All issues were resolved

Contact Information

Company	Representative	Position	Email
Rice	Nguyen Bao Trong	General Director	kn@ricewallet.io
CyStack	Vo Huyen Nhi	Sales Manager	nhivh@cystack.net

Pentesters

Fullname	Role	Email address
Nguyen Huu Trung	Head of Security	trungnh@cystack.net
Nguyen Trung Huy Son	Pentester	
Vu Hai Dang	Pentester	
Ha Minh Chau	Pentester	
Nguyen Van Huy	Pentester	
Nguyen Ba Anh Tuan	Pentester	

Executive Summary

From 11/02/2022 to 01/03/2022, Rice engaged CyStack to evaluate the security posture of its infrastructure compared to current industry best practices. This security audit includes two parts that were conducted simultaneously. The first part was source code review, performed by CyStack, and strictly followed *OWASP Code Review Guide*. The second part was external penetration testing, conducted by CyStack. Test cases for penetration testing were based on the *NIST SP 800-115 Technical Guide to Information Security Testing and Assessment*, *OWASP Testing Guide (v4)*, and customized testing frameworks from CyStack.

CyStack's security assessment for Rice focused on evidence, which confirmed that Rice Wallet properly functions as a non-custodial cryptocurrency wallet, as well as security issues that are typical to DApps. The assessment emphasized remediation over analyzing exploitability, including issues reported by tools. This means that less time was spent determining how specific security flaws might be exploited and more time identifying as many possible security issues and associated remediation as time allowed. The audit results also included a cursory review of dependent libraries and recommendations for improving software assurance practices at Rice Wallet.

1.1 Key Findings

CyStack did not find any proves that indicates vulnerable usage and storage of users' wallet private keys in Rice Wallet, nor any critical severity issues that would undermine the security of confidential transactions. CyStack identified a number of misconfigurations and coding errors that could result in informational leakage, unused server takeover, or potentially circumvent business logic data validation. Additionally, some third-party open source library dependencies were identified as being out of date. Key findings from the engagement included:

- Security misconfigurations that might lead to informational disclosure or server takeover.
- Improper data validation, which attackers can leverage to undermine referral code system rules.
- The usage of deprecated third-party library TrustCore for iOS.

1.2 Limitations

Because of the quantity of static and dynamic analysis diagnostics, some findings were not fully analyzed during the assessment and some security vulnerabilities in third-party open source library dependencies might have not been discovered. Some effort was redirected to propose detailed remediations to the development team to ensure that the repairs would be made before the initial release of the product.

1.3 Assessment Components

Source Code Review

Source code contains the most detailed information about an application. Source code review allows security researchers to understand thoroughly how an application operates and performs. Researchers then can search for design flaws and security vulnerabilities in the application.

The safety and security assessment for application source code includes automated and manual tests. For automated tests, static code analysis tools are used to identify dead code, unsafe coding patterns and the usage of libraries or plugins with publicly known vulnerabilities. Automated tests also search for the the existence of hard-coded sensitive information such as passwords, database connection strings, private keys for third-party services, etc.

Manual tests focus on analyzing the implementation of the application's operational logic and functional components, in order to detect critical vulnerabilities, which are possibly related to user input validation, unsafe database querying, unsafe file handling, etc. or business logic flows. People who perform manual tests are security researchers.

External Penetration Test

An external penetration test emulates the role of an attacker attempting to gain access to an internal network without internal resources or inside knowledge. A CyStack engineer attempts to gather sensitive information through open-source intelligence (OSINT), including employee information, historical breached passwords, and more that can be leveraged against external systems to gain internal network access. The engineer also performs scanning and enumeration to identify potential vulnerabilities in hopes of exploitation.

Scope

Assessment	Details	Type
Source Code Review	Wallet Blockchain / Wallet Android	Source code
Source Code Review	Wallet Blockchain / Wallet IOS	Source code
External Penetration Test	https://www.getpostman.com/collections/5a0d247b8016c729d5d8	API
External Penetration Test	Rice Wallet Android Mobile App (Staging)	Android
External Penetration Test	Rice Wallet iOS Mobile App (Staging)	iOS

Scope Exclusions

Per client request, CyStack did not perform any Denial of Service attacks during testing.

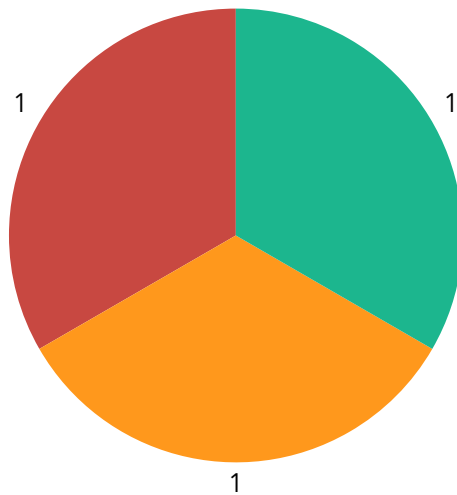
Client Allowances

Rice did not provide any allowances to assist the testing.

Dashboard

Maintaining a healthy security posture requires constant review and refinement of existing security processes. Running a CyStack Pentest allows Rice's internal security team to not only uncover specific vulnerabilities but gain a better understanding of the current security threat landscape.

Vulnerabilities by severities



Legend



Vulnerabilities by assets

Rice Wallet Android Mobile Application	2	<div><div></div><div></div></div>
Others (*.ricewallet.io)	1	<div><div></div></div>

Vulnerabilities by CWE

Broken Access Control (CWE-723)	1	<div><div></div></div>
Injection (CWE-929)	1	<div><div></div></div>
Sensitive Data Exposure (CWE-934)	1	<div><div></div></div>

Vulnerabilities by OWASP Top 10














OWASP Top 10 Category	Test result	Findings
A1 - Broken Access Control		1
A2 - Cryptographic Failures		0
A3 - Injection		1
A4 - Insecure Design		0
A5 - Security Misconfiguration		1
A6 - Vulnerable and Outdated Components		0
A7 - Identification and Authentication Failures		0
A8 - Software and Data Integrity Failures		0
A9 - Security Logging and Monitoring Failures		0
A10 - Server-Side Request Forgery		0

Table of vulnerabilities

ID	Status	Vulnerability	Severity
#rice-001	Fixed	Remote code execution due to incomplete WordPress installation	
#rice-002	Fixed	Visible detailed error	
#rice-003	Fixed	Improper validation for referral code rewarding system	

Code Review Details

3.1 Steps to Conduct

CyStack analyzed the source code and mobile applications using a variety of static and dynamic analysis tools. Specifically, CyStack:

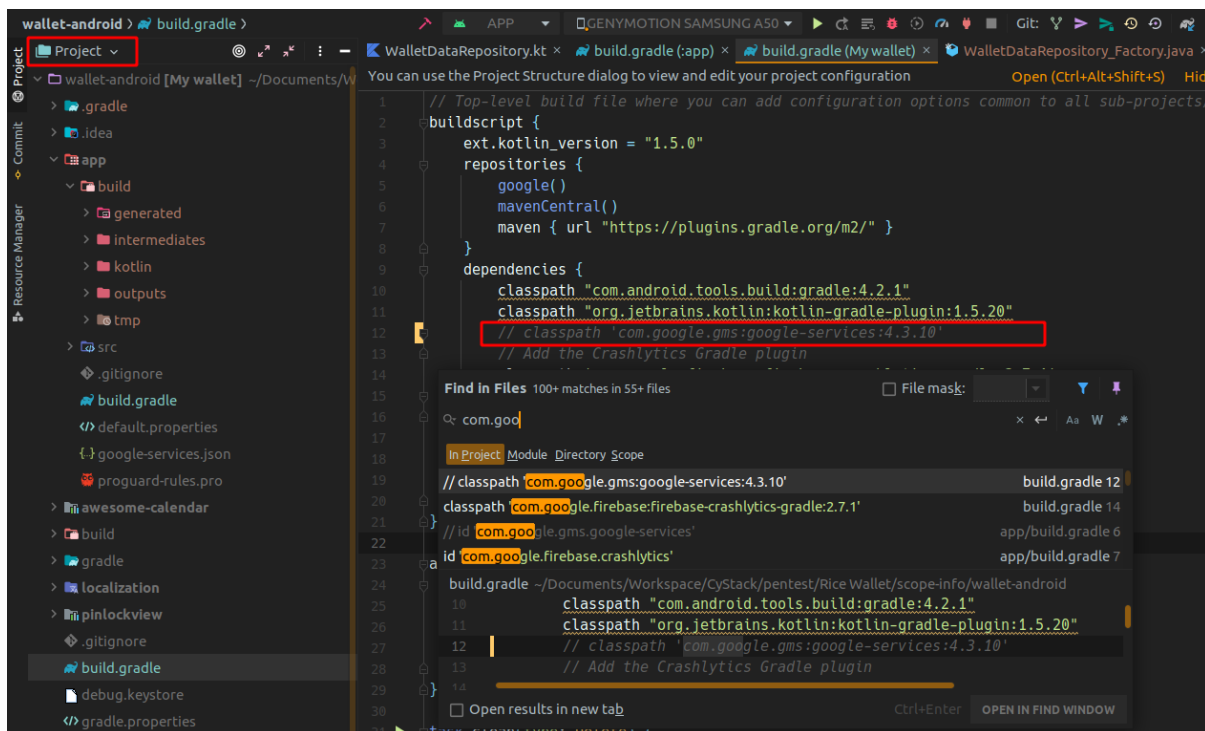
1. Statically analyzed Rice Wallet Android and iOS applications with [MobSF](#). The results are detailed in the chapter Asset Metadata, Rice Wallet iOS mobile application and Rice Wallet Android mobile application.
2. Statically analyzed Rice Wallet Android and iOS code projects with [Insider](#).
3. Used Android Studio to open the source code for Rice Wallet Android mobile application for both static and dynamic analysis.
4. Statically analyzed the source code for Rice Wallet Android mobile application with Android Lint.
5. Dynamically analyzed Rice Wallet for Android by rebuilding it from the given source code and debugging it with an emulator from Android Studio.
6. Used XCode to open the source code for Rice Wallet iOS mobile application for both static and dynamic analysis.
7. Statically analyzed the source code for Rice Wallet iOS mobile application with [SWAN](#).
8. Dynamically analyzed Rice Wallet for iOS by rebuilding it from the given source code and debugging it with a simulator from XCode.

3.2 Results

After reviewing the source code and pentesting mobile applications of Rice Wallet, it can be concluded that no evidence on vulnerable usage and storage of users' wallet private keys in Rice Wallet, nor any critical severity issues that affect confidential transactions were found.

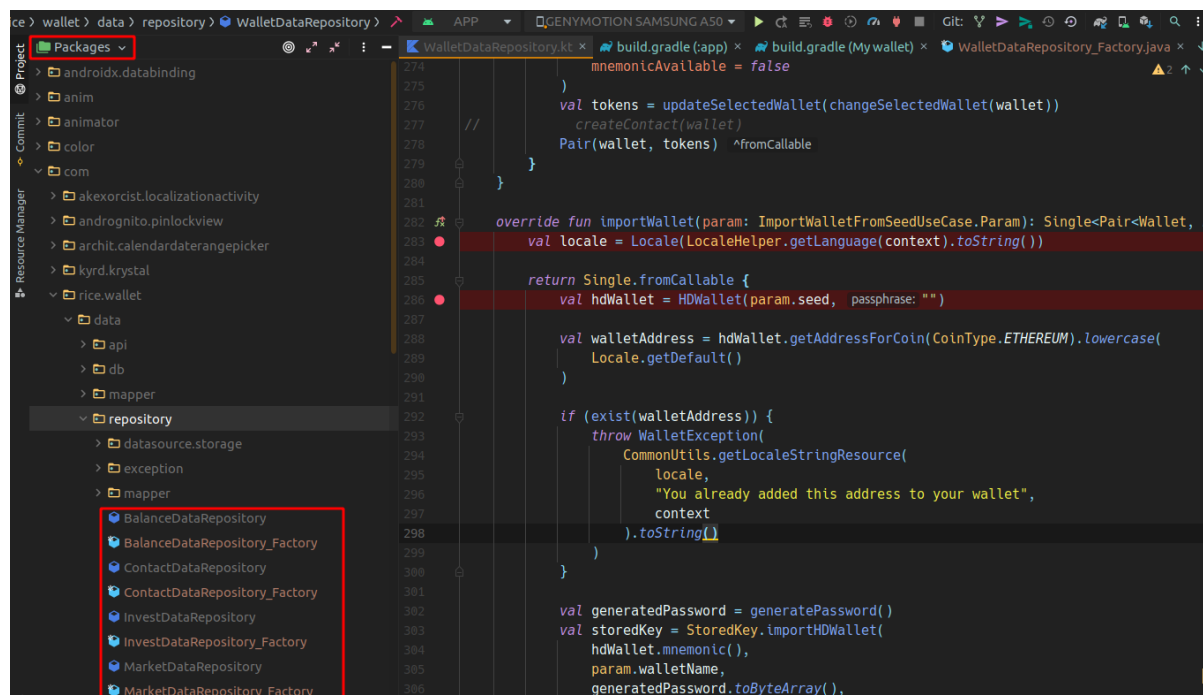
Android code and application analysis

In order to successfully rebuild Rice Wallet application from the source code, a dependency on `com.google.gms` was required to be commented to build the application.

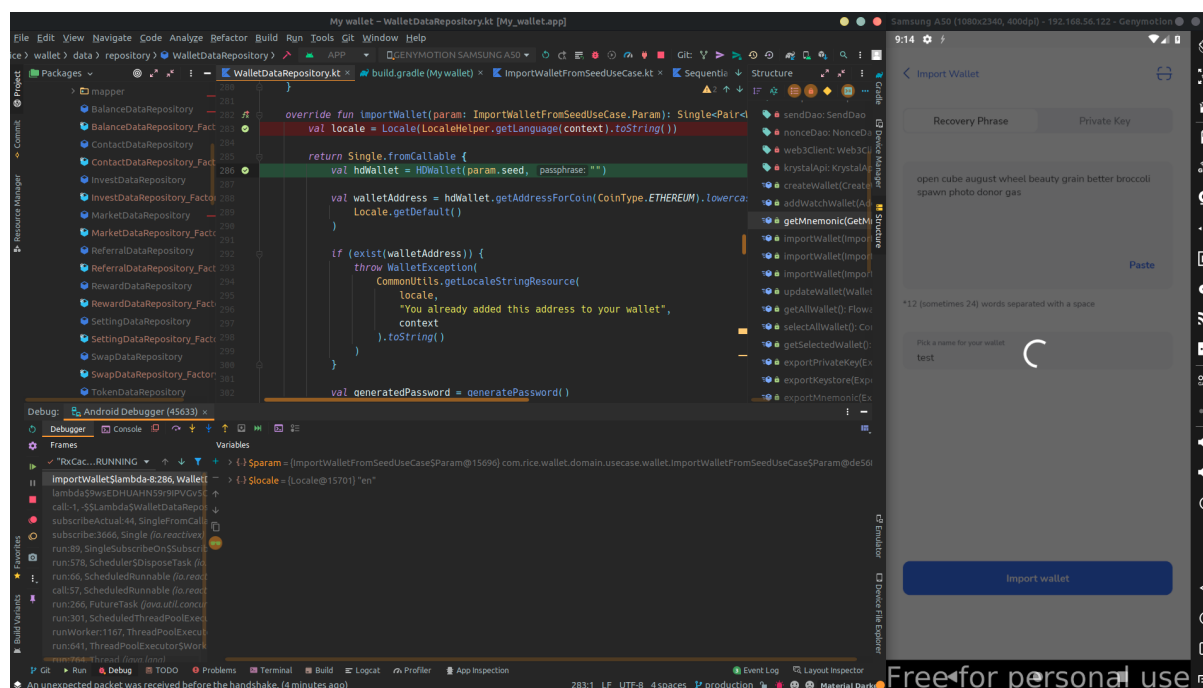


CyStack, by then, debugged the application with Attach mode on Android Studio.

The main codebase for Rice Wallet Android mobile application was found in the folder **com/rice/wallet/data/repository**:



CyStack team then further investigated the code in this folder. When debugging Rice Wallet for Android, the application called for function **importWallet(param: ImportWalletFromSeedUseCase.Param)** to import existed cryptocurrency wallet address. It is shown from this function that the Recovery Phrase for a user's wallet was stored locally by using an object called **HDWallet**, implemented in **com.trustwallet** library. **com.wallet** is a third-party library from [TrustWallet](#). It is a safe and secure cross-platform, cross-blockchain wallet library that are widely used by the DeFi community.



iOS code and application analysis

CyStack rebuilt Rice Wallet iOS mobile application with XCode and debugged with a simulator provided by XCode.

From the specification file called Podfile, which describes the dependencies of the targets of one or more Xcode projects, CyStack identified the use of similar cross-platform, cross-blockchain wallet libraries from Trust Wallet, which are [TrustCore](#) and [TrustKeystore](#).

Podfile 1.15 KB

```
1 platform :ios, '11.0'
2 inhibit_all_warnings!
3 source 'https://github.com/CocoaPods/Specs.git'
4
5 target 'Rice-iOS' do
6   use_frameworks!
7
8   pod 'BigInt', '~> 3.1.0'
9   pod 'JSONRPCKit', '~> 3.0.0' #:git=> 'https://github.com/bricklife/JSONRPCKit.git'
10  pod 'APIKit', '~> 3.2.1'
11  pod 'SVProgressHUD'
12  pod 'KeychainSwift', '~> 13.0.0'
13  pod 'SwiftLint', '~> 0.29.4'
14  pod 'RealmSwift', '~> 10'
15  pod 'Moya', '~> 10.0.1'
16  pod 'JavaScriptKit', '~> 1.0.0'
17  pod 'CryptoSwift', '~> 1.4.1'
18  pod 'Kingfisher', '~> 5.13'
19  pod 'TrustCore', '~> 0.0.7'
20  pod 'TrustKeystore', '~> 0.4.2'
21  pod 'SAMKeychain', '~> 1.5.3'
22  pod 'IQKeyboardManager', '~> 6.5'
23  pod 'SwiftMessages', '~> 9.0.6'
24  pod 'Starscream', '~> 3.1'
25  pod 'Firebase/Analytics'
26  pod 'Firebase/Crashlytics'
27  pod 'Firebase/Messaging'
28  pod 'Charts'
29  pod 'KeychainAccess'
30  pod 'PromiseKit'
31  pod 'EasyNotificationBadge'
32  pod 'Hero'
33
```

CyStack further investigated on these two libraries, and there were found proves on the two git repositories that these libraries are no longer supported by Trust Wallet or deprecated. As for TrustCore, it is recommended by the developers to use [TrustWalletCore](#) instead.

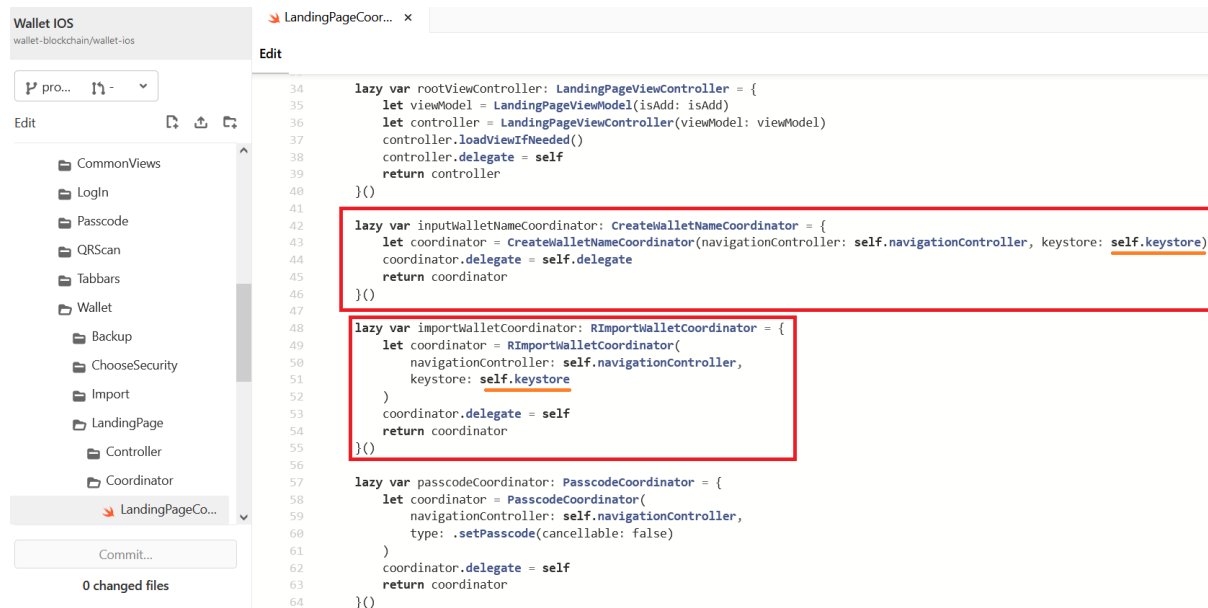
The main codebase for Rice Wallet iOS mobile application was found in the folder **Rice-iOS/Rice-iOS**:

production
wallet-ios / Rice-iOS / Rice-iOS / +

Name	Last commit
..	
Configurations	Add Polygon chain
Coordinators	Format string
Extensions	Add rewards at Summary
Models	Add rewards at Summary
Networking	Add rewards at Summary
Screens	Add reload rewards
Storage	Add Polygon chain
Utilities	Add Polygon chain

CyStack team then reviewed the code in this folder. The landing page logic is implemented in the file `Rice-iOS/Rice-iOS/Screens/Wallet/LandingPage/Coordinator/LandingPageCoordinator.swift`. To create a new wallet, a **self.inputWalletNameCoordinator**, which is a `CreateWalletNameCoordinator` object, would be called, and to import an existed wallet a **importWalletCoordinator**, which is a `RImportWalletCoordinator` object, would be called.

```
extension LandingPageCoordinator: LandingPageViewControllerDelegate {
    func landingPageViewController(_ controller: LandingPageViewController, run event: LandingPageViewEvent) {
        switch event {
        case .openCreateWallet:
            self.inputWalletNameCoordinator.start()
        case .openImportWallet:
            self.navigationController.dismiss(animated: true, completion: nil)
            self.importWalletCoordinator.start()
        case .back:
            self.stop()
        }
    }
}
```



In the files, implemented these coordinators, the object **Wallet** from **Rice-iOS/Extensions/EtherClient/Wallet.swift** and **WalletStorage** from **Rice-iOS/Rice-iOS/Storage/WalletStorage.swift** are called. These are the objects for usage and storage cryptocurrency wallets defined by mentioned Trust Wallet libraries.

Recommendations

Based on the results of this assessment, CyStack has the following high-level key recommendations:

Key recommendations	
Issues	<p>After the source code review and penetration testing processes for Rice Wallet mobile applications, CyStack confirmed that users' wallet private keys are safely handled and stored by Rice Wallet, using a reliable third-party library provided Trust Wallet. In addition, CyStack discovered three vulnerabilities in Rice Wallet system, in which two are from Rice Wallet Android Mobile Application, and one with critical severity from a domain belonging to Rice.</p> <p>These vulnerabilities are all resolved by Rice Wallet team. Deprecated libraries used in Rice Wallet are updated to the recommended by vendors.</p>
Recommendations	<ul style="list-style-type: none">• Review the listed vulnerabilities to anticipate where similar risks may occur, thereby deploying early remediation plans.• Check the dependencies and, if possible, have plans on dropping deprecated libraries and moving to those more up-to-date.
References	<ul style="list-style-type: none">• https://developer.android.com/training/articles/security-tips• https://github.com/Checkmarx/Kotlin-SCP• https://securecode.wiki/docs/lang/kotlin/• https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html• https://securecode.wiki/docs/lang/swift/

Asset Metadata

1. stg-api.ricewallet.io

Basic Information

Type	Website
FQDN	stg-api.ricewallet.io
IP address	104.21.10.78
Operating system	N/A
Last location	US

Open Ports

Port	Protocol	Service	Product	Version
80	tcp	http	Cloudflare http proxy	
443	tcp	ssl/http	Cloudflare http proxy	
2052	tcp	http	Cloudflare http proxy	
2053	tcp	ssl/http	nginx	
2082	tcp	http	Cloudflare http proxy	
2083	tcp	ssl/http	nginx	
2086	tcp	http	Cloudflare http proxy	
2087	tcp	ssl/http	nginx	
2095	tcp	http	Cloudflare http proxy	
2096	tcp	ssl/http	nginx	
8080	tcp	http	Cloudflare http proxy	
8443	tcp	ssl/http	Cloudflare http proxy	
8880	tcp	http	Cloudflare http proxy	

Installed Applications

Program	Vendor	Installed on	Size	Version	URL location	Port
Cloudflare	Cloudflare, Inc.				https://stg-api.ricewallet.io	443
Cloudflare Network Error Logging	Cloudflare, Inc.				https://stg-api.ricewallet.io	443
Node.js	OpenJS Foundation				https://stg-api.ricewallet.io	443
Express	OpenJS Foundation				https://stg-api.ricewallet.io	443

2. Rice Wallet iOS mobile application

Basic Information

Type	Mobile Application
Operating system	iOS
Filename	Rice Wallet 1.0.3.ipa
Size	35.8 MB
MD5	e851e18d13581f2360afd7178105bcb9
SHA1	dc3f4ddf71466c1e0d8670c4667aed836f05d200
SHA256	e2aae1659f8e5f1f867a4a15b4f08ea9c19bf959018acf97be7f887be74f7491
Application name	Rice Wallet
Application type	Swift
Identifier	com.rice.wallet
SDK name	iphoneos15.2
Application version	1.0.3
Build	10
Platform version	15.2
Minimum OS version	11.0

Application permissions

Permission	Severity	Definition	Description
NSCameraUsageDescription	Dangerous	Access camera	Scan QR Code to import wallet and connect Dapps
NSLocationWhenInUseUsageDescription	Dangerous	Access location information when app is in the foreground	Access location to improve Rice Wallet's service
NSPhotoLibraryUsageDescription	Dangerous	Access the user's photo library	Sending QR Code to your contacts
NSFaceIDUsageDescription	Dangerous	Access the user's photo library	Sending QR Code to your contacts

3. Rice Wallet Android mobile application

Basic Information

Type	Mobile Application
Operating system	Android
File name	rice-stg-debug.apk
Size	55.35 MB
MD5	86b081026e36141988a8f47be7b02774
SHA1	d8dfc0c6f8cfb2600b1e1767a5a0623601f46ffe
SHA256	82937912a15dd88fe436ac5b1c69311a710d798ac0fa2a08082776cb3e5e62e8
Application name	RICE-Stg
Package name	com.rice.wallet
Main activity	com.rice.wallet.presentation.splash.SplashActivity
Target SDK	31
Android version	1.0
Android version code	2

Application certificates

v1 signature	True
v2 signature	True
v3 signature	False
Subject	CN=Android Debug, O=Android, C=US
Signature Algorithm	rsassa_pkcs1v15
Valid From	2021-08-03 08:47:52+00:00
Valid To	2051-07-27 08:47:52+00:00
Issuer	CN=Android Debug, O=Android, C=US
Serial Number	0x1
Hash Algorithm	sha1
MD5	fa9639a46cf37b1a453ab5de4d9680c1
SHA1	a0c22900938d2af4a842b8debae56d1bd663151a

SHA256 ceafe449278e52c4c2199d9df7d9ec961a77ca1a2174546a31786bcc57c37167

SHA512 e1a87fd6ec428ce9efd4ca7d116380c7c41f8f7091a0bdaaa0897bf6e38930041dd31f8ccb878a0b9dae141fc5f4e8563b0bafc19a03ebcb6a5c5711050fa77c

Public Key Algorithm rsa

Bit Size 2048

Fingerprint 337cd1e6d1f5dc00829a360e0351a1c20a0421d0bc2ac0ee3d81de20ce29e27a

Application permissions

Permission	Severity	Definition	Description
android.permission.CAMERA	Dangerous	Take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
android.permission.READ_EXTERNAL_STORAGE	Dangerous	Read external storage contents	Allows an application to read from external storage.
android.permission.WRITE_EXTERNAL_STORAGE	Dangerous	Read/modify/delete external storage contents	Allows an application to write to external storage.
android.permission.ACCESS_NETWORK_STATE	Normal	View network status	Allows an application to view the status of all networks.
android.permission.ACCESS_WIFI_STATE	Normal	View Wi-Fi status	Allows an application to view the information about the status of Wi-Fi.
android.permission.FOREGROUND_SERVICE	Normal		Allows a regular application to use Service.startForeground.
android.permission.INTERNET	Normal	Full Internet access	Allows an application to create network sockets.

android.permission.RECEIVE_BOOT_COMPLETED	Normal	Automatically start at boot	Allows an application to start itself as soon as the system has finished booting. This can make it take longer to start the phone and allow the application to slow down the overall phone by always running.
android.permission.USE_BIOMETRIC	Normal		Allows an app to use device supported biometric modalities.
android.permission.USE_FINGERPRINT	Normal	Allow use of fingerprint	This constant was deprecated in API level 28. Applications should request USE_BIOMETRIC instead.
android.permission.WAKE_LOCK	Normal	Prevent phone from sleeping	Allows an application to prevent the phone from going to sleep.
com.google.android.c2dm.permission.RECEIVE	Signature	C2DM permissions	Permission for cloud to device messaging.
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE	Unknown	Unknown permission	Unknown permission from android reference.

Vulnerability Details

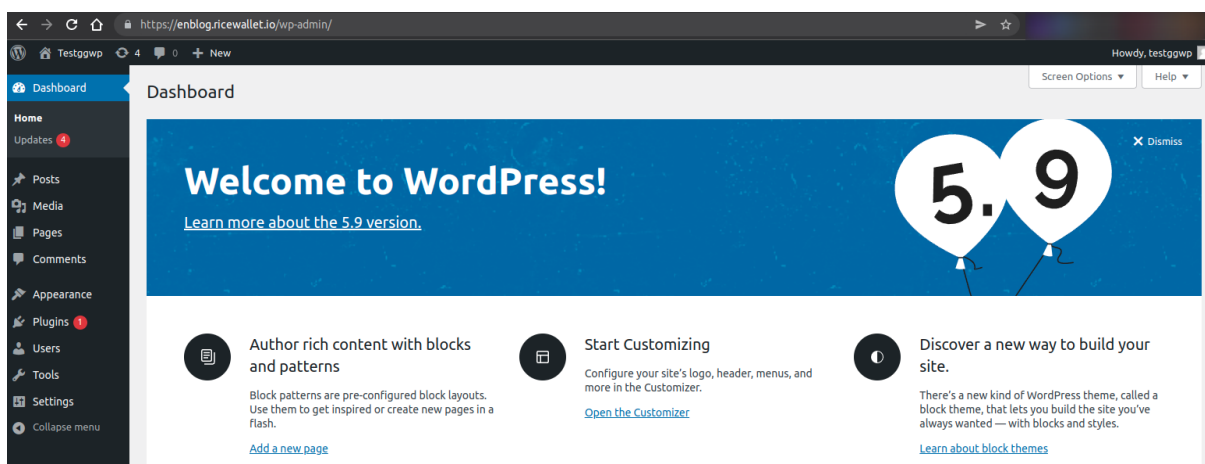
1. Remote code execution due to incomplete WordPress installation

ID	#rice-001
Category	Injection
Description	The website https://enblog.ricewallet.io/ has not been completed installed with WordPress. Unauthenticated attackers could leverage this incomplete installation to access WordPress' settings, create an administration account and complete the installation. With WordPress administrator privileges, attackers may edit source code of any installed plugin on the WordPress site. Then, they create a webshell, execute arbitrary code and successfully take the WordPress server under their control.
Severity	CRITICAL
CVSS 3.0 base score	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:H (9.4)
Target	https://enblog.ricewallet.io/
Status	Fixed
Reference	OWASP A3 - Injection
Remediation	Remove the current installation and reinstall WordPress or shut down this unused site.

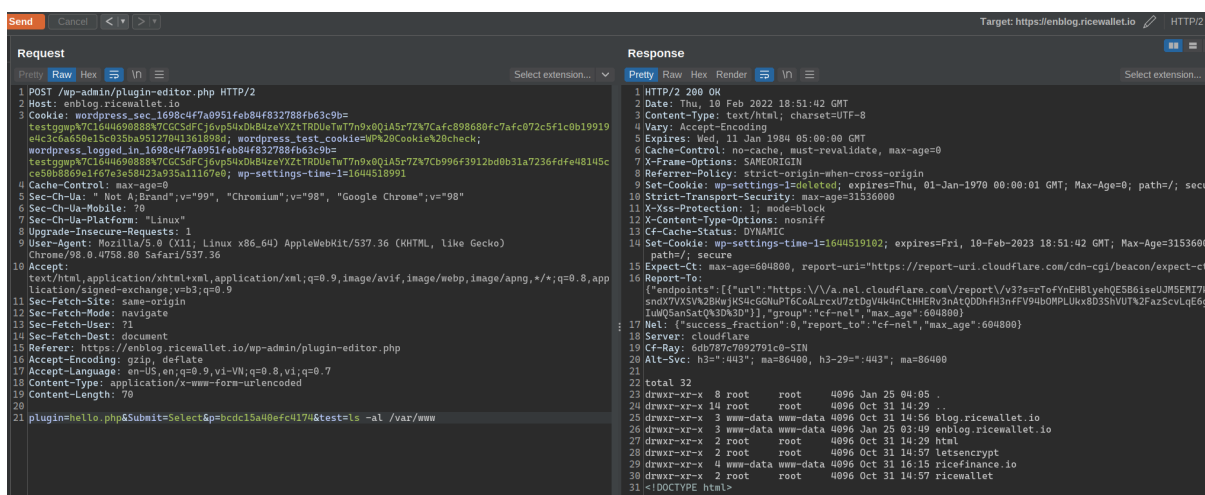
Step to reproduce

1. Create a WordPress administration account and complete the installation:

2. Access WordPress administration page:



3. Create a webshell and execute arbitrary code:



2. Visible detailed error

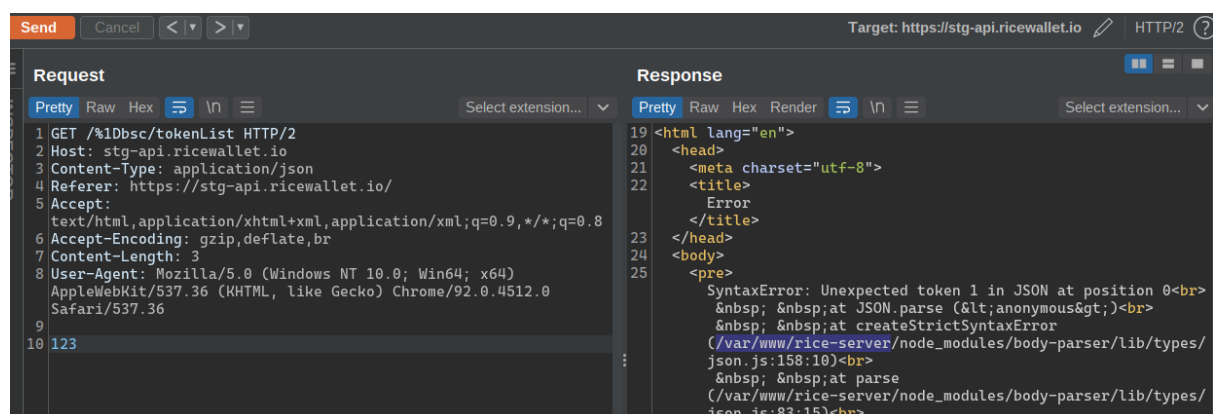
ID	#rice-002
Category	Sensitive Data Exposure
Description	Stack Trace is enabled on the website https://stg-api.ricewallet.io at the endpoint /%1Dbsc/tokenList, which leads to an application full path exposure.
Severity	INFO
CVSS 3.0 base score	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N (0.0)
Target	https://stg-api.ricewallet.io/%1Dbsc/tokenList
Status	Fixed
Reference	OWASP A5 – Security Misconfiguration
Remediation	Unenable Stack Trace or use custom error page for error handling.

Step to reproduce

1. The request that triggers application full path disclosure:

```
GET /%1Dbsc/tokenList HTTP/2
Host: stg-api.ricewallet.io
Content-Type: application/json
Referer: https://stg-api.ricewallet.io/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,br
Content-Length: 3
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/92.0.4512.0 Safari/537.36
```

2. Response with the application full path:



3. Improper validation for referral code rewarding system

ID	#rice-003
Category	Broken Access Control
Description	At the endpoint https://stg-api.ricewallet.io/referral/add-ref , variables address and device_id have not been carefully handled and validated. Attackers may leverage this bug to control and increase the number of invited users.
Severity	MEDIUM
CVSS 3.0 base score	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N (6.5)
Target	https://stg-api.ricewallet.io/referral/add-ref
Status	Fixed
Reference	OWASP A1 - Broken Access Control
Remediation	Validate variables address and device_id before process responses.

Step to reproduce

1. Send a request to the endpoint `/referral/add-ref` with arbitrary values for **device_id** and **address**. The referral code is from the application on testing device.

The screenshot shows a REST client interface with a POST request to `/referral/add-ref` on the target `https://stg-api.ricewallet.io`. The request body is a JSON object with the following fields:

```

{
  "device_id": "161462349",
  "address": "0xaebd161462349",
  "code": "KZH9PFK0"
}

```

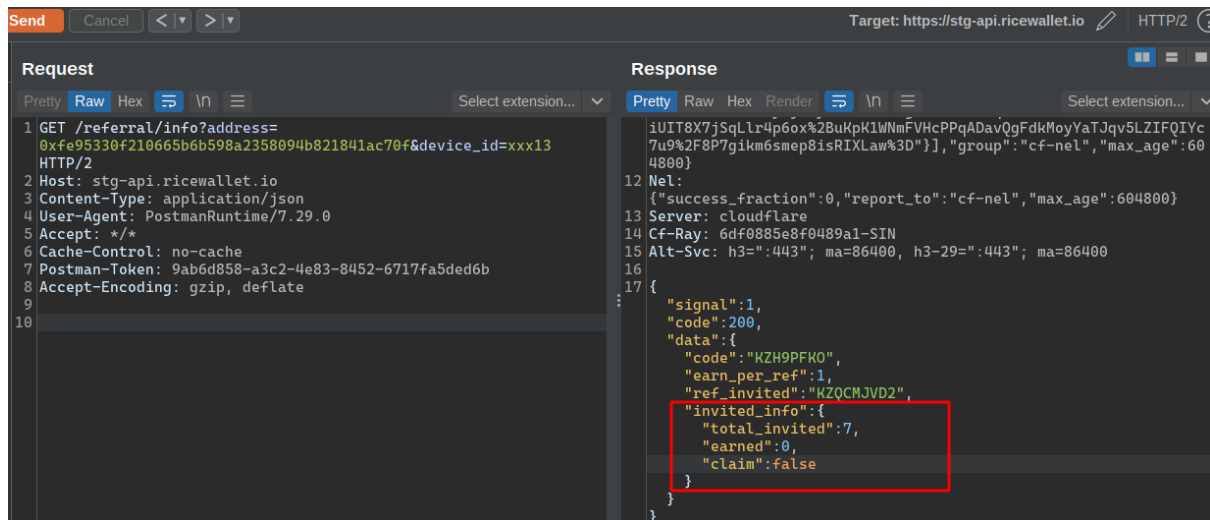
The response is a 200 OK status with a JSON body containing a signal, code, and data object:

```

{
  "signal": 1,
  "code": 200,
  "data": {
    "ref_invited": "KZH9PFK0"
  }
}

```

2. Check the number of invited users by the response. It has been increased:



Appendix

Appendix A - Vulnerability Severity Ratings

Severity	CVSS 3.0 score range	Definition
CRITICAL	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
HIGH	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
MEDIUM	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
LOW	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
INFO	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Appendix B - Vulnerability Categories

CyStack uses CWE (Common Weakness Enumeration) for the vulnerability categorization. Common Weakness Enumeration (CWE) is a community-developed list of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.

CWE categories used by CyStack are listed in the following table:

CWE ID	Name
CWE-16	Security Misconfiguration
CWE-77, CWE-259	Insecure OS Firmware
CWE-79	Cross-Site Scripting (XSS)
CWE-310	Broken Cryptography
CWE-311, CWE-319	Insecure Data Transport
CWE-352	Cross-Site Request Forgery (CSRF)
CWE-359	Privacy Concerns
CWE-400	Application Level Denial Of Service (DoS)
CWE-601	Unvalidated Redirects And Forwards
CWE-693	Lack Of Binary Hardening
CWE-723	Broken Access Control
CWE-729, CWE-922	Insecure Data Storage
CWE-919	Mobile Security Misconfiguration
CWE-929	Injection
CWE-930	Broken Authentication And Session Management
CWE-934	Sensitive Data Exposure
CWE-937	Using Components With Known Vulnerabilities

Appendix C - Security Assessment Based On OWASP Web Security Testing Guide v4.2

Test ID	Test name	Status
WSTG-INFO	Information Gathering	Pass
WSTG-INFO-01	Conduct Search Engine Discovery and Reconnaissance for Information Leakage	Pass
WSTG-INFO-02	Fingerprint Web Server	Pass
WSTG-INFO-03	Review Webserver Metafiles for Information Leakage	Pass
WSTG-INFO-04	Enumerate Applications on Webserver	Pass
WSTG-INFO-05	Review Webpage Content for Information Leakage	Pass
WSTG-INFO-06	Identify Application Entry Points	Pass
WSTG-INFO-07	Map Execution Paths Through Application	Pass
WSTG-INFO-08	Fingerprint Web Application Framework	Pass
WSTG-INFO-09	Fingerprint Web Application	Pass
WSTG-INFO-10	Map Application Architecture	Pass
WSTG-CONF	Configuration and Deploy Management Testing	Pass
WSTG-CONF-01	Test Network Infrastructure Configuration	Pass
WSTG-CONF-02	Test Application Platform Configuration	Pass
WSTG-CONF-03	Test File Extensions Handling for Sensitive Information	Pass
WSTG-CONF-04	Review Old Backup and Unreferenced Files for Sensitive Information	Pass
WSTG-CONF-05	Enumerate Infrastructure and Application Admin Interfaces	Pass
WSTG-CONF-06	Test HTTP Methods	Pass
WSTG-CONF-07	Test HTTP Strict Transport Security	Pass
WSTG-CONF-08	Test RIA Cross Domain Policy	Pass
WSTG-CONF-09	Test File Permission	Pass
WSTG-CONF-10	Test for Subdomain Takeover	Pass
WSTG-CONF-11	Test Cloud Storage	Pass

WSTG-CONF-12	Testing for Content Security Policy	Pass
WSTG-IDNT	Identity Management Testing	Pass
WSTG-IDNT-01	Test Role Definitions	Pass
WSTG-IDNT-02	Test User Registration Process	Pass
WSTG-IDNT-03	Test Account Provisioning Process	Pass
WSTG-IDNT-04	Testing for Account Enumeration and Guessable User Account	Pass
WSTG-IDNT-05	Testing for Weak or Unenforced Username Policy	Pass
WSTG-ATHN	Authentication Testing	Pass
WSTG-ATHN-01	Testing for Credentials Transported over an Encrypted Channel	Pass
WSTG-ATHN-02	Testing for Default Credentials	Pass
WSTG-ATHN-03	Testing for Weak Lock Out Mechanism	Pass
WSTG-ATHN-04	Testing for Bypassing Authentication Schema	Pass
WSTG-ATHN-05	Testing for Vulnerable Remember Password	Pass
WSTG-ATHN-06	Testing for Browser Cache Weakness	Pass
WSTG-ATHN-07	Testing for Weak Password Policy	Pass
WSTG-ATHN-08	Testing for Weak Security Question Answer	Pass
WSTG-ATHN-09	Testing for Weak Password Change or Reset Functionalities	Pass
WSTG-ATHN-10	Testing for Weaker Authentication in Alternative Channel	Pass
WSTG-ATHZ	Authorization Testing	Pass
WSTG-ATHZ-01	Testing Directory Traversal File Include	Pass
WSTG-ATHZ-02	Testing for Bypassing Authorization Schema	Pass
WSTG-ATHZ-03	Testing for Privilege Escalation	Pass
WSTG-ATHZ-04	Testing for Insecure Direct Object References	Pass
WSTG-SESS	Session Management Testing	Pass
WSTG-SESS-01	Testing for Session Management Schema	Pass
WSTG-SESS-02	Testing for Cookies Attributes	Pass
WSTG-SESS-03	Testing for Session Fixation	Pass

WSTG-SESS-04	Testing for Exposed Session Variables	Pass
WSTG-SESS-05	Testing for Cross Site Request Forgery	Pass
WSTG-SESS-06	Testing for Logout Functionality	Pass
WSTG-SESS-07	Testing Session Timeout	Pass
WSTG-SESS-08	Testing for Session Puzzling	Pass
WSTG-SESS-09	Testing for Session Hijacking	Pass
WSTG-SESS-10	Testing JSON Web Tokens	Pass
WSTG-INPV	Input Validation Testing	Pass
WSTG-INPV-01	Testing for Reflected Cross Site Scripting	Pass
WSTG-INPV-02	Testing for Stored Cross Site Scripting	Pass
WSTG-INPV-03	Testing for HTTP Verb Tampering	Pass
WSTG-INPV-04	Testing for HTTP Parameter pollution	Pass
WSTG-INPV-05	Testing for SQL Injection	Pass
WSTG-INPV-06	Testing for LDAP Injection	Pass
WSTG-INPV-07	Testing for XML Injection	Pass
WSTG-INPV-08	Testing for SSI Injection	Pass
WSTG-INPV-09	Testing for XPath Injection	Pass
WSTG-INPV-10	Testing for IMAP SMTP Injection	Pass
WSTG-INPV-11	Testing for Code Injection	Pass
WSTG-INPV-12	Testing for Command Injection	Pass
WSTG-INPV-13	Testing for Format String Injection	Pass
WSTG-INPV-14	Testing for Incubated Vulnerabilities	Pass
WSTG-INPV-15	Testing for HTTP Splitting Smuggling	Pass
WSTG-INPV-16	Testing for HTTP Incoming Requests	Pass
WSTG-INPV-17	Testing for Host Header Injection	Pass
WSTG-INPV-18	Testing for Server-side Template Injection	Pass
WSTG-INPV-19	Testing for Server-Side Request Forgery	Pass
WSTG-ERRH	Error Handling	Pass

WSTG-ERRH-01	Testing for Improper Error Handling	Pass
WSTG-ERRH-02	Testing for Stack Traces	Pass
WSTG-CRYP	Cryptography	Pass
WSTG-CRYP-01	Testing for Weak Transport Layer Security	Pass
WSTG-CRYP-02	Testing for Padding Oracle	Pass
WSTG-CRYP-03	Testing for Sensitive Information Sent Via Unencrypted Channels	Pass
WSTG-CRYP-04	Testing for Weak Encryption	Pass
WSTG-BUSLOGIC	Business Logic Testing	Pass
WSTG-BUSL-01	Test Business Logic Data Validation	Pass
WSTG-BUSL-02	Test Ability to Forge Requests	Pass
WSTG-BUSL-03	Test Integrity Checks	Pass
WSTG-BUSL-04	Test for Process Timing	Pass
WSTG-BUSL-05	Test Number of Times a Function Can be Used Limits	Pass
WSTG-BUSL-06	Testing for the Circumvention of Work Flows	Pass
WSTG-BUSL-07	Test Defenses Against Application Misuse	Pass
WSTG-BUSL-08	Test Upload of Unexpected File Types	Pass
WSTG-BUSL-09	Test Upload of Malicious Files	Pass
WSTG-CLIENT	Client-side Testing	Pass
WSTG-CLNT-01	Testing for DOM based Cross Site Scripting	Pass
WSTG-CLNT-02	Testing for JavaScript Execution	Pass
WSTG-CLNT-03	Testing for HTML Injection	Pass
WSTG-CLNT-04	Testing for Client-side URL Redirect	Pass
WSTG-CLNT-05	Testing for CSS Injection	Pass
WSTG-CLNT-06	Testing for Client-side Resource Manipulation	Pass
WSTG-CLNT-07	Test Cross Origin Resource Sharing	Pass
WSTG-CLNT-08	Testing for Cross Site Flashing	Pass
WSTG-CLNT-09	Testing for Clickjacking	Pass
WSTG-CLNT-10	Testing WebSockets	Pass

WSTG-CLNT-11	Test Web Messaging	Pass
WSTG-CLNT-12	Test Browser Storage	Pass
WSTG-CLNT-13	Testing for Cross Site Script Inclusion	Pass
WSTG-APIT	API Testing	Pass
WSTG-APIT-01	Testing GraphQL	Pass

LEGEND

Pass: Requirement is applicable to Web application and implemented according to best practices.

Fail: Requirement is applicable to Web application but not fulfilled.

Appendix D - Security Assessment Based On OWASP Mobile Security Testing Guide v1.2

Mobile Application Security Requirements

Test ID	Test name	Status
	Architecture, design and threat modelling	
MSTG-ARCH-1	All app components are identified and known to be needed.	Pass
MSTG-ARCH-2	Security controls are never enforced only on the client side, but on the respective remote endpoints.	Pass
MSTG-ARCH-3	A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture.	Pass
MSTG-ARCH-4	Data considered sensitive in the context of the mobile app is clearly identified.	Pass
MSTG-ARCH-5	All app components are defined in terms of the business functions and/or security functions they provide.	N/A
MSTG-ARCH-6	A threat model for the mobile app and the associated remote services has been produced that identifies potential threats and countermeasures.	N/A
MSTG-ARCH-7	All security controls have a centralized implementation.	N/A
MSTG-ARCH-8	There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57.	N/A
MSTG-ARCH-9	A mechanism for enforcing updates of the mobile app exists.	N/A
MSTG-ARCH-10	Security is addressed within all parts of the software development lifecycle.	N/A
MSTG-ARCH-11	A responsible disclosure policy is in place and effectively applied.	N/A
MSTG-ARCH-12	The app should comply with privacy laws and regulations.	Pass
	Data Storage and Privacy	

MSTG-STORAGE-1	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.	Pass
MSTG-STORAGE-2	No sensitive data should be stored outside of the app container or system credential storage facilities.	Pass
MSTG-STORAGE-3	No sensitive data is written to application logs.	Pass
MSTG-STORAGE-4	No sensitive data is shared with third parties unless it is a necessary part of the architecture.	Pass
MSTG-STORAGE-5	The keyboard cache is disabled on text inputs that process sensitive data.	Pass
MSTG-STORAGE-6	No sensitive data is exposed via IPC mechanisms.	Pass
MSTG-STORAGE-7	No sensitive data, such as passwords or pins, is exposed through the user interface.	Pass
MSTG-STORAGE-8	No sensitive data is included in backups generated by the mobile operating system.	N/A
MSTG-STORAGE-9	The app removes sensitive data from views when moved to the background.	N/A
MSTG-STORAGE-10	The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.	N/A
MSTG-STORAGE-11	The app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode.	N/A
MSTG-STORAGE-12	The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.	N/A
MSTG-STORAGE-13	No sensitive data should be stored locally on the mobile device. Instead, data should be retrieved from a remote endpoint when needed and only be kept in memory.	N/A
MSTG-STORAGE-14	If sensitive data is still required to be stored locally, it should be encrypted using a key derived from hardware backed storage which requires authentication.	N/A
MSTG-STORAGE-15	The app's local storage should be wiped after an excessive number of failed authentication attempts.	N/A
	Cryptography	

MSTG-CRYPTO-1	The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.	Pass
MSTG-CRYPTO-2	The app uses proven implementations of cryptographic primitives.	Pass
MSTG-CRYPTO-3	The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.	Pass
MSTG-CRYPTO-4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	Pass
MSTG-CRYPTO-5	The app doesn't re-use the same cryptographic key for multiple purposes.	Pass
MSTG-CRYPTO-6	All random values are generated using a sufficiently secure random number generator.	Pass
	Authentication and Session Management	
MSTG-AUTH-1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	Pass
MSTG-AUTH-2	If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials.	Pass
MSTG-AUTH-3	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.	Pass
MSTG-AUTH-4	The remote endpoint terminates the existing session when the user logs out.	Pass
MSTG-AUTH-5	A password policy exists and is enforced at the remote endpoint.	Pass
MSTG-AUTH-6	The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.	Pass
MSTG-AUTH-7	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.	Pass

MSTG-AUTH-8	Biometric authentication, if any, is not event-bound (i.e. using an API that simply returns "true" or "false"). Instead, it is based on unlocking the keychain/keystore.	N/A
MSTG-AUTH-9	A second factor of authentication exists at the remote endpoint and the 2FA requirement is consistently enforced.	N/A
MSTG-AUTH-10	Sensitive transactions require step-up authentication.	N/A
MSTG-AUTH-11	The app informs the user of all sensitive activities with their account. Users are able to view a list of devices, view contextual information (IP address, location, etc.), and to block specific devices.	N/A
MSTG-AUTH-12	Authorization models should be defined and enforced at the remote endpoint.	Pass
	Network Communication	
MSTG-NETWORK-1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	Pass
MSTG-NETWORK-2	The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.	Pass
MSTG-NETWORK-3	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.	Pass
MSTG-NETWORK-4	The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.	N/A
MSTG-NETWORK-5	The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.	N/A
MSTG-NETWORK-6	The app only depends on up-to-date connectivity and security libraries.	N/A
	Platform Interaction	
MSTG-PLATFORM-1	The app only requests the minimum set of permissions necessary.	Pass

MSTG-PLATFORM-2	All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.	Pass
MSTG-PLATFORM-3	The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.	Pass
MSTG-PLATFORM-4	The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.	Pass
MSTG-PLATFORM-5	JavaScript is disabled in WebViews unless explicitly required.	Pass
MSTG-PLATFORM-6	WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.	Pass
MSTG-PLATFORM-7	If native methods of the app are exposed to a WebView, verify that the WebView only renders JavaScript contained within the app package.	Pass
MSTG-PLATFORM-8	Object deserialization, if any, is implemented using safe serialization APIs.	Pass
MSTG-PLATFORM-9	The app protects itself against screen overlay attacks. (Android only)	N/A
MSTG-PLATFORM-10	A WebView's cache, storage, and loaded resources (JavaScript, etc.) should be cleared before the WebView is destroyed.	N/A
MSTG-PLATFORM-11	Verify that the app prevents usage of custom third-party keyboards whenever sensitive data is entered.	N/A
	Code Quality and Build Settings	
MSTG-CODE-1	The app is signed and provisioned with a valid certificate, of which the private key is properly protected.	Pass
MSTG-CODE-2	The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).	Pass
MSTG-CODE-3	Debugging symbols have been removed from native binaries.	Pass

MSTG-CODE-4	Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.	Pass
MSTG-CODE-5	All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.	Pass
MSTG-CODE-6	The app catches and handles possible exceptions.	Pass
MSTG-CODE-7	Error handling logic in security controls denies access by default.	Pass
MSTG-CODE-8	In unmanaged code, memory is allocated, freed and used securely.	Pass
MSTG-CODE-9	Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.	Pass

Resiliency against Reverse Engineering

Test ID	Test name	Status
	Impede Dynamic Analysis and Tampering	
MSTG-RESILIENCE-1	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.	N/A
MSTG-RESILIENCE-2	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.	N/A
MSTG-RESILIENCE-3	The app detects, and responds to, tampering with executable files and critical data within its own sandbox.	N/A
MSTG-RESILIENCE-4	The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.	N/A
MSTG-RESILIENCE-5	The app detects, and responds to, being run in an emulator.	N/A
MSTG-RESILIENCE-6	The app detects, and responds to, tampering the code and data in its own memory space.	N/A
MSTG-RESILIENCE-7	The app implements multiple mechanisms in each defense category (8.1 to 8.6). Note that resiliency scales with the amount, diversity of the originality of the mechanisms used.	N/A
MSTG-RESILIENCE-8	The detection mechanisms trigger responses of different types, including delayed and stealthy responses.	N/A
MSTG-RESILIENCE-9	Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.	N/A
	Device Binding	
MSTG-RESILIENCE-10	The app implements a 'device binding' functionality using a device fingerprint derived from multiple properties unique to the device.	N/A
	Impede Comprehension	

MSTG-RESILIENCE-11	All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.	N/A
MSTG-RESILIENCE-12	If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.	N/A
Impede Eavesdropping		
MSTG-RESILIENCE-13	As a defense in depth, next to having solid hardening of the communicating parties, application level payload encryption can be applied to further impede eavesdropping.	N/A

LEGEND

Pass: Requirement is applicable to Mobile application and implemented according to best practices.

Fail: Requirement is applicable to Mobile application but not fulfilled.

N/A: Requirement is not applicable to Mobile application.