// Web and Mobile Pentest

12.30.2024 - 01.22.2025

SSP Wallet, Relay and Key

InFlux Technologies

HALBORN

SSP Wallet, Relay and Key - InFlux Technologies

Prepared by: HALBORN

Last Updated 03/06/2025

Date of Engagement by: December 30th, 2024 - January 22nd, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
35	0	3	15	17	0

TABLE OF CONTENTS

8.12 Hal-10 - mobile - weak password policy
8.13 Hal-33 - android - exposure of sensitive data through clipboard
8.14 Hal-25 - be - potential risk due to third-party iframe
8.15 Hal-30 - api - outdated versions of tls supported
8.16 Hal-14 - api - cacheable https response
8.17 Hal-15 - ios - lack of jailbreak detection mechanism
8.18 Hal-16 - android - lack of root detection mechanism
8.19 Hal-12 - ios - certificate pinning bypass
8.20 Hal-13 - android - certificate pinning bypass
8.21 Hal-35 - ios - lack of anti-tampering and anti-hooking mechanisms
8.22 Hal-34 - android - lack of anti-tampering and anti-hooking mechanisms
8.23 Hal-28 - be - verbose logging in extension
8.24 Hal-06 - be - lack of password complexity and password policy
8.25 Hal-11 - ios - background screen caching
8.26 Hal-19 - android - background screen caching
8.27 Hal-24 - android - transaction data exposed in logs
8.28 Hal-21 - ios - application allows screenshots
8.29 Hal-20 - android - application allows screenshots
8.30 Hal-23 - api - lack of rate limitation on ssp relay endpoints
8.31 Hal-27 - api - lack of data sanitization and validation of limits
8.32 Hal-05 - ios - insecure accessibility attributes in keychain storage
8.33 Hal-22 - be - potential risk of sensitive data exposure through clipboard
8.34 Hal-03 - be - unrestrictive content-security-policy (csp)
8.35 Hal-32 - android - risk of overlay attack

1. Introduction

`InFlux Technologies` engaged Halborn to conduct a security assessment of their applications. The security assessment was scoped to their browser extension, their respective underlying API, and mobile applications (Android and iOS). Halborn was provided access to the application and its respective source code to conduct security testing using tools to scan, detect, and validate possible vulnerabilities found in the application and report the findings at the end of the engagement.

2. Assessment Summary

The team at Halborn was provided a timeline for the engagement and assigned a full-time security engineer to verify 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 security assessment identified several vulnerabilities across the application ecosystem, affecting extension, backend, API, and mobile platforms. High-risk issues include the exposure of sensitive mnemonic phrases in memory and insecure storage of PINs on iOS, which could lead to unauthorized access and compromise user accounts. Hardcoded secrets found in public repositories further heighten the risk of exploitation. Medium-risk findings included biometric authentication bypass on Android and iOS, insecure trust in device-level biometrics, and the potential exposure of sensitive data due to insecure clipboard usage. Vulnerable third-party dependencies and lack of rate limitation on critical API endpoints were also observed, which could result in denial-of-service attacks or sensitive data leaks.

Other significant issues included weak password policies, unrestricted Content-Security-Policy configurations, and the use of insecure accessibility attributes in iOS keychain storage. The insecure handling of third-party iframes and risks associated with overlay attacks on Android were noted as medium to low risk but require attention to ensure robust protection against phishing and impersonation. Lower-severity issues, such as the support for outdated TLS versions, cacheable HTTPS responses, verbose logging, and lack of anti-tampering and anti-hooking mechanisms, were also documented. While these may not pose an immediate threat, addressing them will enhance the overall security posture. Mitigating these vulnerabilities will improve the overall security posture of the applications.

The **InFlux Technologies** team addressed most of the identified issues, with one partially resolved, some marked as risk accepted, and others scheduled for resolution in future builds of the application.

3. Test Approach And Methodology

Halborn followed Whitebox and Blackbox methodology as per the scope and performed a combination of manual and automated security testing with both 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 assessment methodology covered included but was not limited to a range of phases and employed various tools.

- Mapping Content and Functionality of Applications
- Application Logic Flaw
- Reverse Engineering the applications
- Access Handling
- Authentication/Authorization Flaw
- Transaction Flow
- Rate Limitations Test
- Input Handling
- Source Code Review
- Mobile Specific Vulnerabilities
- Fuzzing of all input parameter

4. Scope

URL/API:

`https://relay.ssp.runonflux.io/`

Binaries:

- **Android APK:** `Version 1.5.1`
- **iOS IPA:** `Version 1.5.1 Build 92`
- **Browser Extension:** `v1.8.4`

5. 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

6. SCOPE

FILES AND REPOSITORY

(a) Repository: ssp-relay
(b) Assessed Commit ID: 98a7d85
(c) Items in scope:

Out-of-Scope:

FILES AND REPOSITORY

(a) Repository: ssp-key
(b) Assessed Commit ID: e346264
(c) Items in scope:

Out-of-Scope:

FILES AND REPOSITORY

(a) Repository: ssp-wallet
(b) Assessed Commit ID: 4f8c894
(c) Items in scope:

Out-of-Scope:

REMEDIATION COMMIT ID:

- https://github.com/RunOnFlux/ssp-wallet/pull/376
- 9ae59d2
- https://github.com/RunOnFlux/ssp-key/pull/76
- d2d5a0e
- a17bd25
- dc10902
- https://github.com/RunOnFlux/ssp-key/pull/79

Out-of-Scope: New features/implementations after the remediation commit IDs.

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	3	15	17	0

IMPACT X LIKELIHOOD

		HAL-26 HAL-04 HAL-29		
	HAL-17 HAL-18 HAL-31 HAL-07 HAL-08 HAL-10 HAL-33 HAL-25	HAL-01		
	HAL-02 HAL-09	HAL-23 HAL-27 HAL-05 HAL-22 HAL-03 HAL-32		
	HAL-30 HAL-14 HAL-15 HAL-16 HAL-12 HAL-13 HAL-35 HAL-34 HAL-28 HAL-06 HAL-11 HAL-19			

	HAL-24 HAL-21 HAL-20			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-02 - BE - DEPENDENCIES SHOULD BE PINNED TO EXACT VERSIONS	LOW	SOLVED - 01/31/2025
HAL-09 - IOS - PERSISTENT KEYCHAIN DATA	LOW	SOLVED - 02/25/2025
HAL-26 - BE - MNEMONIC PHRASE EXPOSURE IN MEMORY	HIGH	SOLVED - 02/14/2025
HAL-04 - IOS - INSECURE STORAGE OF PIN	HIGH	SOLVED - 02/19/2025
HAL-29 - HARCODED SECRETS IN GIT HISTORY	HIGH	SOLVED - 01/29/2025
HAL-01 - VULNERABLE THIRD-PARTY DEPENDENCIES	MEDIUM	SOLVED - 02/25/2025
HAL-17 - IOS - BIOMETRIC AUTHENTICATION BYPASS	MEDIUM	SOLVED - 02/17/2025
HAL-18 - ANDROID - FINGERPRINT AUTHENTICATION BYPASS	MEDIUM	SOLVED - 02/17/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-31 - IOS - ALLOWING SENSITIVE DATA TO BE COPIED TO CLIPBOARD	MEDIUM	SOLVED - 02/17/2025
HAL-07 - ANDROID - INSECURE TRUST OF DEVICE-LEVEL BIOMETRIC AUTHENTICATION	MEDIUM	SOLVED - 02/17/2025
HAL-08 - IOS - INSECURE TRUST OF DEVICE-LEVEL BIOMETRIC AUTHENTICATION	MEDIUM	SOLVED - 02/13/2025
HAL-10 - MOBILE - WEAK PASSWORD POLICY	MEDIUM	SOLVED - 02/14/2025
HAL-33 - ANDROID - EXPOSURE OF SENSITIVE DATA THROUGH CLIPBOARD	MEDIUM	SOLVED - 02/17/2025
HAL-25 - BE - POTENTIAL RISK DUE TO THIRD-PARTY IFRAME	MEDIUM	SOLVED - 01/15/2025
HAL-30 - API - OUTDATED VERSIONS OF TLS SUPPORTED	LOW	SOLVED - 01/29/2025
HAL-14 - API - CACHEABLE HTTPS RESPONSE	LOW	SOLVED - 01/29/2025
HAL-15 - IOS - LACK OF JAILBREAK DETECTION MECHANISM	LOW	SOLVED - 02/19/2025
HAL-16 - ANDROID - LACK OF ROOT DETECTION MECHANISM	LOW	SOLVED - 02/19/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-12 - IOS - CERTIFICATE PINNING BYPASS	LOW	FUTURE RELEASE - 02/21/2025
HAL-13 - ANDROID - CERTIFICATE PINNING BYPASS	LOW	FUTURE RELEASE - 02/21/2025
HAL-35 - IOS - LACK OF ANTI-TAMPERING AND ANTI-HOOKING MECHANISMS	LOW	FUTURE RELEASE - 02/21/2025
HAL-34 - ANDROID - LACK OF ANTI-TAMPERING AND ANTI-HOOKING MECHANISMS	LOW	FUTURE RELEASE - 02/21/2025
HAL-28 - BE - VERBOSE LOGGING IN EXTENSION	LOW	SOLVED - 01/31/2025
HAL-06 - BE - LACK OF PASSWORD COMPLEXITY AND PASSWORD POLICY	LOW	SOLVED - 01/31/2025
HAL-11 - IOS - BACKGROUND SCREEN CACHING	LOW	SOLVED - 02/19/2025
HAL-19 - ANDROID - BACKGROUND SCREEN CACHING	LOW	SOLVED - 02/17/2025
HAL-24 - ANDROID - TRANSACTION DATA EXPOSED IN LOGS	LOW	SOLVED - 02/17/2025
HAL-21 - IOS - APPLICATION ALLOWS SCREENSHOTS	LOW	FUTURE RELEASE - 02/19/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-20 - ANDROID - APPLICATION ALLOWS SCREENSHOTS	LOW	SOLVED - 02/17/2025
HAL-23 - API - LACK OF RATE LIMITATION ON SSP RELAY ENDPOINTS	MEDIUM	SOLVED - 01/29/2025
HAL-27 - API - LACK OF DATA SANITIZATION AND VALIDATION OF LIMITS	MEDIUM	SOLVED - 01/29/2025
HAL-05 - IOS - INSECURE ACCESSIBILITY ATTRIBUTES IN KEYCHAIN STORAGE	MEDIUM	SOLVED - 02/19/2025
HAL-22 - BE - POTENTIAL RISK OF SENSITIVE DATA EXPOSURE THROUGH CLIPBOARD	MEDIUM	SOLVED - 02/14/2025
HAL-03 - BE - UNRESTRICTIVE CONTENT-SECURITY-POLICY (CSP)	MEDIUM	SOLVED - 01/30/2025
HAL-32 - ANDROID - RISK OF OVERLAY ATTACK	MEDIUM	PARTIALLY SOLVED - 02/25/2025

8. FINDINGS & TECH DETAILS

8.1 HAL-02 - BE - DEPENDENCIES SHOULD BE PINNED TO EXACT VERSIONS

// LOW

Description

The application contained multiple dependencies that were not pinned to an exact version, but they were set to a supported version (x̂.x.x). This could potentially allow dependency attacks, as seen with the flow of events package with Copay Bitcoin Wallet and `@solana/web3.js` npm package.

Proof of Concept

ssp-wallet-4f8c894a3e9a7058c541848910100bb2624a8a0f > {} package.json > ...

```
  "scripts": {
  },
  "dependencies": {
    "@alchemy/aa-core": "~3.19.0",
    "@metamask/browser-passworder": "~5.0.1",
    "@noble/hashes": "~1.6.1",
    "@reduxjs/toolkit": "~2.3.0",
    "@runonflux/aa-schnorr-multisig-sdk": "~1.0.4",
    "@runonflux/flux-sdk": "~1.0.3",
    "@runonflux/utxo-lib": "~1.0.1",
    "@scure/bip32": "~1.6.0",
    "@scure/bip39": "~1.5.0",
    "@ant-design/icons": "~5.5.1",
    "antd": "~5.22.2",
    "assert": "~2.1.0",
    "axios": "~1.7.8",
    "bchaddrjs": "~0.5.2",
    "bignumber.js": "~9.1.2",
    "buffer": "~6.0.3",
    "crypto-browserify": "~3.12.1",
    "currency-symbol-map": "~5.1.0",
    "events": "~3.3.0",
    "i18next": "~24.0.2",
    "localforage": "~1.10.0",
    "lru-cache": "~11.0.2",
    "patch-package": "~8.0.0",
    "postinstall-postinstall": "~2.1.0",
    "process": "~0.11.10",
    "react": "~18.3.1",
    "react-countdown-circle-timer": "~3.2.1",
    "react-dom": "~18.3.1",
    "react-i18next": "~15.1.2",
    "react-redux": "~9.1.2",
    "react-router": "~7.0.1",
    "react-secure-storage": "~1.3.2",
    "socket.io-client": "~4.8.1",
    "stream-browserify": "~3.0.0",
    "util": "~0.12.5",
    "export-to-csv": "~1.4.0",
    "viem": "~2.21.51"
  },
```

```
  "scripts": {
    "backup": "tsx backup.js"
  },
  "author": "Tadeas Kmenta",
  "license": "MIT",
  "dependencies": {
    "@runonflux/aa-schnorr-multisig-sdk": "~1.0.4",
    "@runonflux/utxo-lib": "~1.0.0",
    "alchemy-sdk": "~3.4.7",
    "apicache": "~1.6.3",
    "axios": "~1.7.7",
    "bchaddrjs": "~0.5.2",
    "bignumber.js": "~9.1.2",
    "bitcoinjs-lib": "~6.1.6",
    "compression": "~1.7.5",
    "config": "~3.3.12",
    "cors": "~2.8.5",
    "express": "~4.21.1",
    "firebase-admin": "~12.7.0",
    "freshdesk-client": "~1.9.1",
    "lru-cache": "~11.0.2",
    "mongodb": "~6.10.0",
    "morgan": "~1.10.0",
    "node-cmd": "~5.0.0",
    "qs": "~6.13.0",
    "socket.io": "~4.8.1",
    "viem": "~2.21.44",
    "zelcorejs": "https://github.com/zelcore-io/zelcorejs"
  },
  "devDependencies": {
    "@eslint/js": "~9.14.0",
    "@types/chai": "~4.3.20",
    "@types/compression": "~1.7.5",
    "@types/cors": "~2.8.17",
    "@types/express": "~4.17.21",
    "@types/expect": "~24.3.2",
    "@types/mocha": "~10.0.9",
    "@types/morgan": "~1.9.9",
    "chai": "~4.5.0",
    "eslint": "~9.14.0",
```

```
  },
  "dependencies": {
    "@alchemy/aa-core": "~3.19.0",
    "@notifee/react-native": "~9.1.3",
    "@react-native-clipboard/clipboard": "~1.15.0",
    "@react-native-firebase/app": "~21.6.1",
    "@react-native-firebase/messaging": "~21.6.1",
    "@react-native-picker/picker": "~2.10.0",
    "@react-navigation/native": "~6.1.18",
    "@react-navigation/stack": "~6.4.1",
    "@reduxjs/toolkit": "~2.3.0",
    "@runonflux/aa-schnorr-multisig-sdk": "~1.0.4",
    "@runonflux/react-native-step-indicator": "~1.0.0",
    "@runonflux/utxo-lib": "~1.0.1",
    "@scure/bip32": "~1.6.0",
    "@scure/bip39": "~1.5.0",
    "abitype": "~1.0.6",
    "assert": "~2.1.0",
    "axios": "~1.7.8",
    "bchaddrjs": "~0.5.2",
    "bignumber.js": "~9.1.2",
    "buffer": "~6.0.3",
    "crypto-js": "~4.2.0",
    "events": "~3.3.0",
    "fastestsmallesttextencoderdecoder": "~1.0.22",
    "i18next": "~24.0.2",
    "postinstall-postinstall": "~2.1.0",
    "process": "~0.11.10",
    "react": "~18.3.1",
    "react-i18next": "~15.1.2",
    "react-native": "~0.75.4",
    "react-native-biometrics": "~3.0.1",
    "react-native-camera-kit": "~14.1.0",
    "react-native-device-info": "~14.0.1",
    "react-native-encrypted-storage": "~4.0.3",
    "react-native-gesture-handler": "~2.21.2",
    "react-native-get-random-values": "~1.11.0",
    "react-native-keyboard-aware-scroll-view": "~0.9.5",
    "react-native-mmkv": "~2.12.2",
    "react-native-permissions": "~5.2.0",
    "react-native-popup-menu": "~0.16.1",
    "react-native-quick-base64": "~2.1.2"
```

Score

Impact: 3

Likelihood: 2

Recommendation

Pinning dependencies to an exact version (=x.x.x) is recommended to reduce the risk of inadvertently introducing a malicious version of a dependency in the future. This helps in mitigating supply chain attack risks, ensuring that updates are controlled and vetted before implementation.

Remediation

SOLVED: The **InFlux Technologies team** resolved the issue by pinning the dependencies.
Commits:

- **SSP Wallet:** `3693a45b56309c09df3dbfaf11dc2088a76b635f`
- **SSP Key:** `125106c300835c2d0d25a0d69430093e3eae6171`
- **SSP Relay:** `d8d64fc43f0f704593428bce7c02ef1e7c5d85f2`

Remediation Hash

https://github.com/RunOnFlux/ssp-wallet/pull/376

8.2 HAL-09 - iOS - Persistent Keychain Data

// LOW

Description

The SSP-Key iOS app stores sensitive information in the iOS Keychain, which persists even after the app is uninstalled and reinstalled. Additionally, the application did not provide a logout functionality, which would allow users to clear sensitive data manually when they no longer need to use the app. Without a logout option, users have no way to explicitly remove stored credentials or tokens. Furthermore, since the keychain data remains intact across uninstalls, this could result in unauthorized access if the device is shared or compromised, as previously stored sensitive data (such as PIN and authentication tokens) can still be accessed after reinstalling the app.

Proof of Concept

- Install the SSP-Key iOS app and set up an account.
- Use a tool such as ios keychain dump to inspect keychain entries.
- Uninstall the SSP-Key app.
- Reinstall the app and observe that the previously stored sensitive keychain entries remain accessible without requiring re-authentication.
- Note that there is no logout button available in the app for users to manually clear sensitive data.

Score

Impact: 3

Likelihood: 2

Recommendation

It is recommended to implement a logout feature that allows users to explicitly clear sensitive data, such as passwords and authentication tokens, from the keychain. This ensures users have control over their sensitive information and can actively secure their account if they suspect unauthorized access. Additionally, sensitive data in the keychain should be stored using secure accessibility attributes, such as `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`, to ensure it is only available when the device is unlocked and a passcode is set. This approach also ensures that sensitive data is deleted when the app is uninstalled. Lastly, avoid storing plaintext passwords directly in the keychain. Instead, store securely hashed and salted values using industry-standard cryptographic algorithms like PBKDF2 or bcrypt.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by implementing a **delete feature**, allowing users to manage and remove their data securely.

8.3 HAL-26 - BE - MNEMONIC PHRASE EXPOSURE IN MEMORY

// HIGH

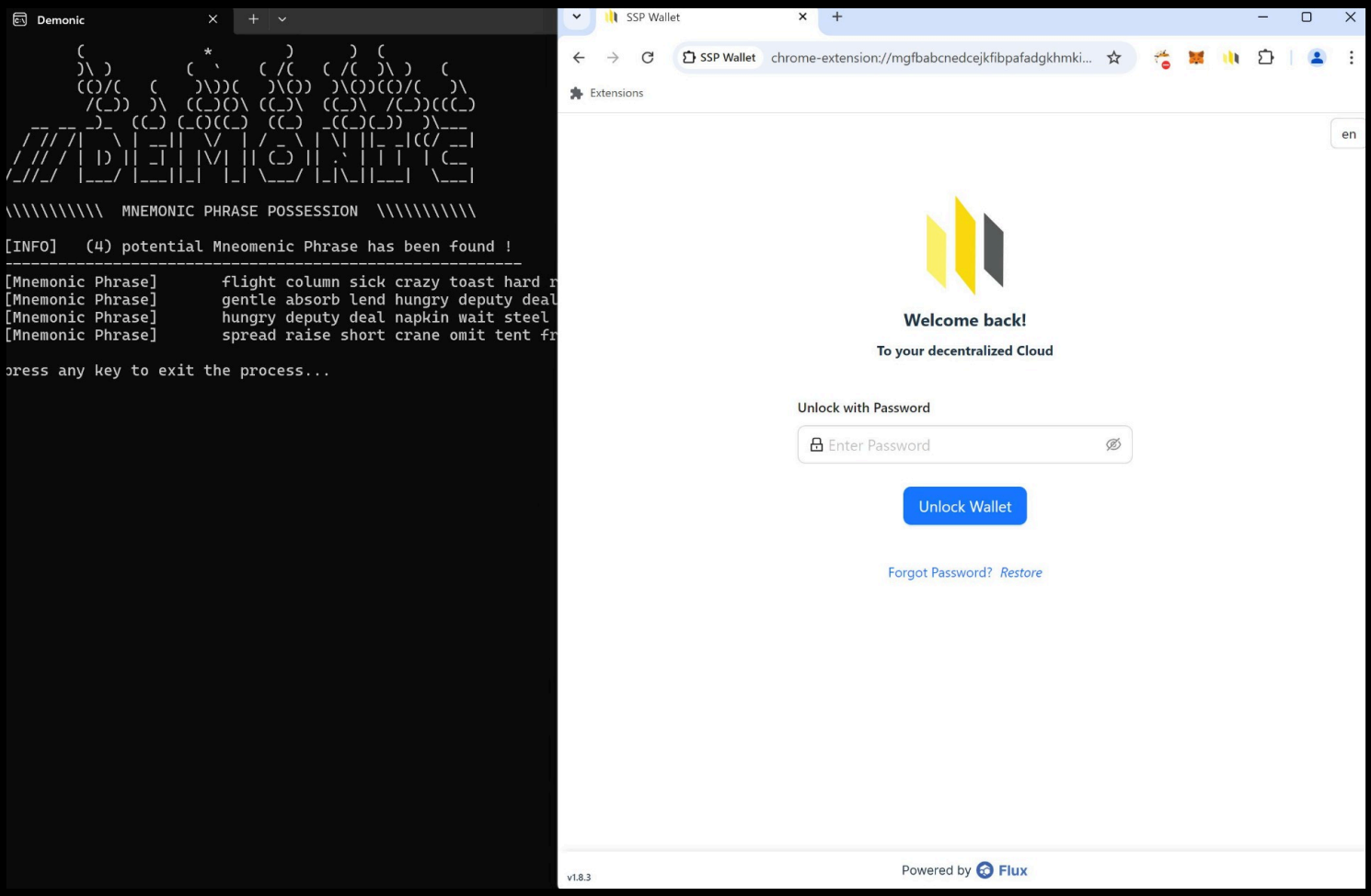
Description

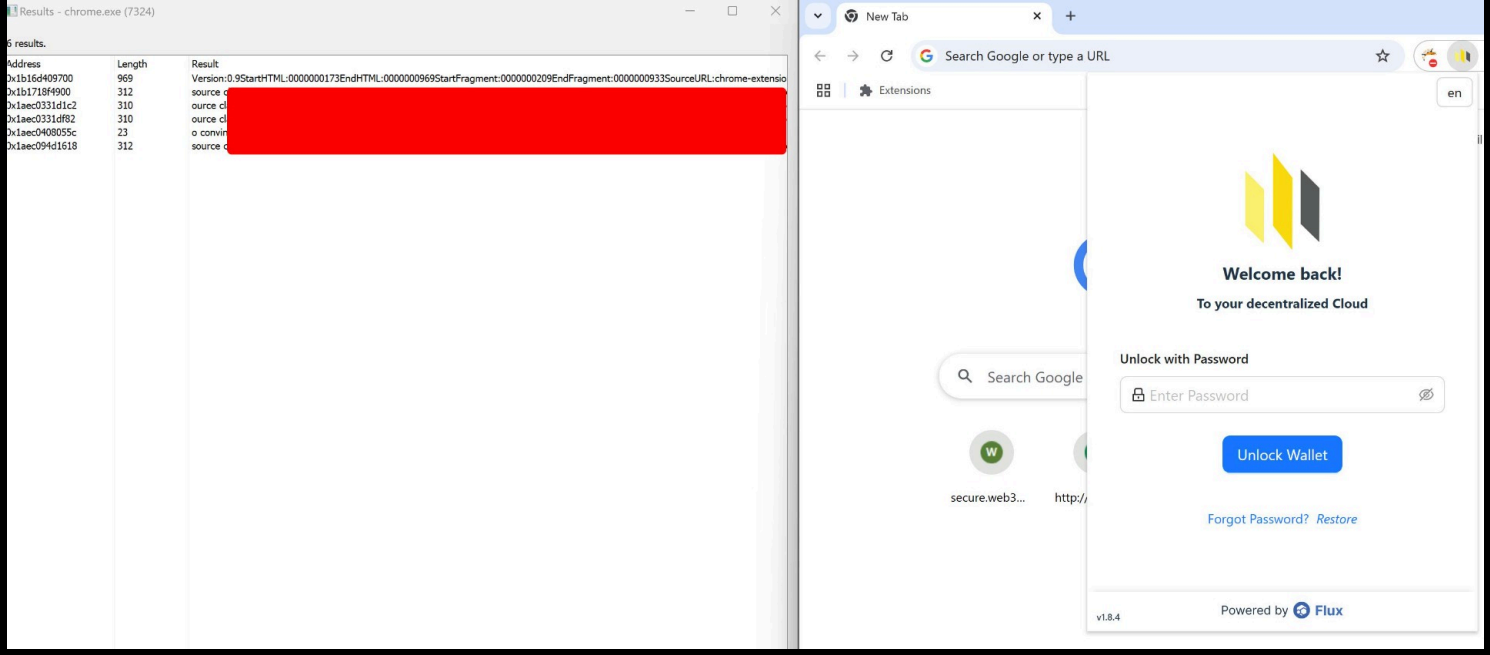
The Mnemonic Phrase of the wallet kept unencrypted in memory, even the wallet was locked. As a result, an attacker with access to the user's machine could exfiltrate the Mnemonic Phrase. It was possible to retrieve the Mnemonic Phrase from memory in these three cases:

- When creating the wallet, it was possible to dump the mnemonic from memory
- When revealing the mnemonic after having logged in, mnemonics stayed in the memory as long as the process running.
- When recovering a wallet by copying the mnemonic and pasting it directly to the browser extension.
- When wallet was in locked state, mnemonics stayed in the memory as long as the process running.

It is crucial to recognize that the mnemonic risk extends beyond the application state; it could also be leaked into memory when the browser displays the mnemonic in clear text and as long as the process running. This potential leakage poses a significant security concern, emphasizing the need for careful handling of such sensitive information within the browser environment.

Proof of Concept





Score

Impact: 5

Likelihood: 3

Recommendation

The identified vulnerability arises from the application's handling of sensitive data in plain text. To mitigate this, the team recommends the following strategies:

- Opt for storing the entropy on disk rather than the mnemonic itself. When the mnemonic is necessary in the code, consider breaking it into multiple variables. Alternatively, obfuscate the original phrase and subsequently dereference the variable holding the original phrase.
- For instances requiring Mnemonic Phrase handling, utilize the obfuscated variable with a function designed to reconstruct the original Mnemonic Phrase exactly at the point of need.
- Ensure that when the wallet is in a locked state, the mnemonic phrase is completely cleared from memory.

For the display and handling of the mnemonic phrase during wallet creation and when revealed to a logged-in user:

- Display the mnemonic phrase using an HTML5 canvas. This technique helps prevent users from copying the phrase, reducing the risk of it being unintentionally stored in memory via the clipboard.
- Limit the ability for users to copy the entire mnemonic from the extension. This approach is essential in minimizing the potential for the mnemonic to be accidentally leaked through the clipboard, thereby enhancing the security of the sensitive information.

Implementing these recommendations would significantly enhance the security of mnemonic phrase handling, reducing the risks associated with its exposure or misuse.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by splitting the seed phrase and private key into multiple parts during copying, which prevents direct exposure. Additional warnings and explanations have also been added to inform users about the associated risks.

Remediation Hash

9ae59d25865e5a5249a850e0a4b37cdc64e5d26a

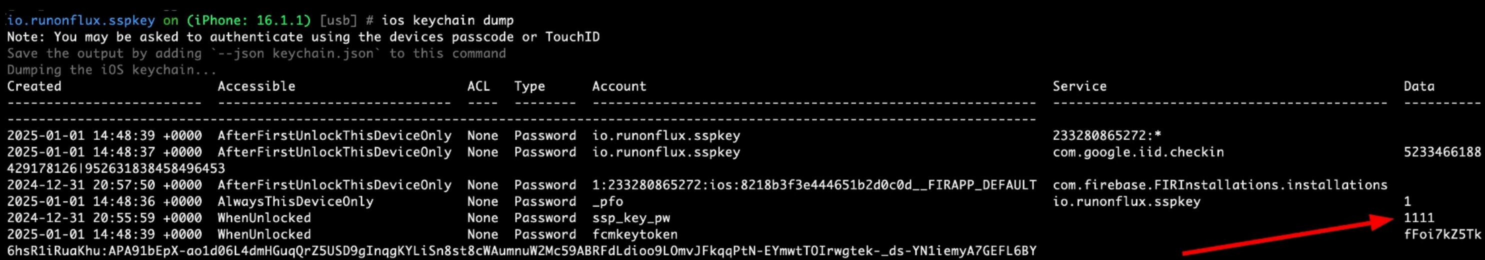
8.4 HAL-04 - IOS - INSECURE STORAGE OF PIN

// HIGH

Description

The SSP-Key iOS application stores a sensitive PIN (`ssp_key_pw`) in **plaintext** within the iOS Keychain. This practice introduces a severe security vulnerability because plaintext storage of authentication data leaves it exposed to direct access if the Keychain is compromised. Since the PIN is neither hashed nor encrypted, an attacker with access to the device or Keychain can extract the PIN in its original form and use it for unauthorized access, impersonation, or decryption of other sensitive data. The accessibility level of `WhenUnlocked` further exacerbates the issue by allowing access to the PIN whenever the device is unlocked, increasing the risk of exploitation.

Proof of Concept



```
io.runonflux.sspkey on (iPhone: 16.1.1) [usb] # ios keychain dump
Note: You may be asked to authenticate using the devices passcode or TouchID
Save the output by adding `--json keychain.json` to this command
Dumping the iOS keychain...
Created                    Accessible                       ACL   Type      Account                                                       Service                                     Data
-------------------------  -------------------------------  ----  --------  ------------------------------------------------------------  ------------------------------------------  ----------
-------------------------------------------------------------------------------------------------------------------------------------------
2025-01-01 14:48:39 +0000  AfterFirstUnlockThisDeviceOnly   None  Password  io.runonflux.sspkey                                           233280865272:*
2025-01-01 14:48:37 +0000  AfterFirstUnlockThisDeviceOnly   None  Password  io.runonflux.sspkey                                           com.google.iid.checkin                      5233466188
429178126|952631838458496453
2024-12-31 20:57:50 +0000  AfterFirstUnlockThisDeviceOnly   None  Password  1:233280865272:ios:8218b3f3e444651b2d0c0d__FIRAPP_DEFAULT     com.firebase.FIRInstallations.installations
2025-01-01 14:48:36 +0000  AlwaysThisDeviceOnly             None  Password  _pfo                                                          io.runonflux.sspkey                         1
2024-12-31 20:55:59 +0000  WhenUnlocked                     None  Password  ssp_key_pw                                                                                                1111
2025-01-01 14:48:39 +0000  WhenUnlocked                     None  Password  fcmkeytoken                                                                                               fFoi7kZ5Tk
6hsR1iRuaKhu:APA91bEpX-ao1d06L4dmHGuqQrZ5USD9gInqgKYLiSn8st8cWAumnuW2Mc59ABRFdLdioo9LOmvJFkqqPtN-EYmwtTOIrwgtek-_ds-YN1iemyA7GEFL6BY
```

Score

Impact: 5

Likelihood: 3

Recommendation

Sensitive authentication data, such as PINs, should never be stored in plaintext. Instead, use a secure, industry-standard cryptographic hashing algorithm to store a salted and hashed version of the PIN. This ensures that even if the Keychain is compromised, the PIN cannot be directly recovered or misused. Additionally, set the Keychain accessibility level to `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` to ensure data is only accessible when the device is unlocked with a secure passcode, adding another layer of protection.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by opting for `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`, ensuring accessibility when the device is unlocked while balancing user experience with security.

Remediation Hash

https://github.com/RunOnFlux/ssp-key/pull/76

8.5 HAL-29 - HARCODED SECRETS IN GIT HISTORY

// HIGH

Description

Sharing code publicly or privately could lead to inadvertently leaving secrets in the code, making them accessible to others. Repositories are frequently cloned and forked into new projects, giving new developers access to their complete history. Any secrets within the repository's history would exist in all new repositories, and if stored in plain text, could lead to potential misuse of the secrets. In the case of a breach or unauthorized access to the source code, it could lead to further exploitation of infrastructure. During the security assessment, it was identified that the application contained a hardcoded **Private Keys** and other service account details. Due to **public repositories**, any unauthorized access to the source code could potentially lead to the misuse of these hardcoded secrets, compromising the security and integrity of the application and its associated infrastructure.

Proof of Concept

- **Identified potential sensitive data in git history here:** https://github.com/RunOnFlux/ssp-relay/commit/d87f1360f9a5854197de4fc5955a916c7ecaabae#diff-977e7becf2c7fbcd18511d0723659f79c7efb748f53bcf2f2d482b8afe013b89

NOTE: Further, test private keys and mnemonics were identified; confirmation was required that these were only used for testing purposes and never used anywhere else.

- https://github.com/RunOnFlux/ssp-wallet/blob/master/tests/lib/wallet.spec.ts
- https://github.com/RunOnFlux/ssp-key/blob/master/tests/lib/wallet.test.ts

Score

Impact: 5

Likelihood: 3

Recommendation

To address the identified risks, the following measures would be appropriate:

- Extracted hardcoded secrets from the source code and the repository's commit history.
- Changed and invalidated exposed secrets, rotate them with new, secure credentials.
- Employed environment variables, secret management solutions, or encryption for secure secret storage and management.

Remediation

SOLVED: The **InFlux Technologies team** confirmed that the credentials identified during the assessment were exclusively for testing purposes and were not used in production.

8.6 HAL-01 - VULNERABLE THIRD-PARTY DEPENDENCIES

// MEDIUM

Description

The scoped repository uses multiple third-party dependencies. Using vulnerable third-party libraries can result in security vulnerabilities in the project that can be exploited by attackers. This can result in data breaches, theft of sensitive information, and other security issues. However, some of them were affected by public-known vulnerabilities that may pose a risk to the global application security level.

Proof of Concept

SSP Relay outdated dependencies:

Path	alchemy-sdk > @ethersproject/contracts > @ethersproject/abi > @ethersproject/hash > @ethersproject/abstract-signer > @ethersproject/abstract-provider > @ethersproject/transactions > @ethersproject/signing-key > elliptic
More info	https://www.npmjs.com/advisories/1101387

low	**Valid ECDSA signatures erroneously rejected in Elliptic**
Package	elliptic
Patched in	>=6.6.0
Dependency of	alchemy-sdk
Path	alchemy-sdk > @ethersproject/wallet > @ethersproject/hdnode > @ethersproject/wordlists > @ethersproject/hash > @ethersproject/abstract-signer > @ethersproject/abstract-provider > @ethersproject/transactions > @ethersproject/signing-key > elliptic
More info	https://www.npmjs.com/advisories/1101387

low	**Valid ECDSA signatures erroneously rejected in Elliptic**
Package	elliptic
Patched in	>=6.6.0
Dependency of	alchemy-sdk
Path	alchemy-sdk > @ethersproject/wallet > @ethersproject/json-wallets > @ethersproject/hdnode > @ethersproject/wordlists > @ethersproject/hash > @ethersproject/abstract-signer > @ethersproject/abstract-provider > @ethersproject/transactions > @ethersproject/signing-key > elliptic
More info	https://www.npmjs.com/advisories/1101387

44 vulnerabilities found - Packages audited: 815
Severity: 41 Low | 1 Moderate | 2 High
✨ Done in 2.25s.

SSP Key outdated dependencies:

```
| high          | ws affected by a DoS when handling a request with many HTTP
|               | headers
| Package       | ws
| Patched in    | >=8.17.1
| Dependency of | @alchemy/aa-core
| Path          | @alchemy/aa-core > viem > ws
| More info     | https://www.npmjs.com/advisories/1098392

| high          | ws affected by a DoS when handling a request with many HTTP
|               | headers
| Package       | ws
| Patched in    | >=8.17.1
| Dependency of | @runonflux/aa-schnorr-multisig-sdk
| Path          | @runonflux/aa-schnorr-multisig-sdk > @alchemy/aa-core > viem
|               | > ws
| More info     | https://www.npmjs.com/advisories/1098392

2 vulnerabilities found - Packages audited: 1341
Severity: 2 High
✨  Done in 2.81s.
```

SSP Wallet outdated dependencies:

```
..00bb2624a8a0f
| Path          | mock-browser > jsdom > tough-cookie
| More info     | https://www.npmjs.com/advisories/1097682

| moderate      | tough-cookie Prototype Pollution vulnerability
| Package       | tough-cookie
| Patched in    | >=4.1.3
| Dependency of | mock-browser
| Path          | mock-browser > jsdom > request > tough-cookie
| More info     | https://www.npmjs.com/advisories/1097682

| high          | ws affected by a DoS when handling a request with many HTTP
|               | headers
| Package       | ws
| Patched in    | >=8.17.1
| Dependency of | @alchemy/aa-core
| Path          | @alchemy/aa-core > viem > ws
| More info     | https://www.npmjs.com/advisories/1098392

| high          | ws affected by a DoS when handling a request with many HTTP
|               | headers
| Package       | ws
| Patched in    | >=8.17.1
| Dependency of | @runonflux/aa-schnorr-multisig-sdk
| Path          | @runonflux/aa-schnorr-multisig-sdk > @alchemy/aa-core > viem
|               | > ws
| More info     | https://www.npmjs.com/advisories/1098392

5 vulnerabilities found - Packages audited: 907
Severity: 3 Moderate | 2 High
✨  Done in 1.63s.
```

Score

Impact: 4

Likelihood: 3

Recommendation

Update all affected packages to its latest version. It is strongly recommended to perform an automated analysis of the dependencies from the birth of the project and if they contain any security issues. Developers should be aware of this and apply any necessary mitigation measures to protect the affected application.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by updating the dependencies to the latest.
Commits:

- **SSP Key:** `b1feed070b9eaeb74dba08b68d41abf9b04e7359`
- **SSP Wallet:** `3693a45b56309c09df3dbfaf11dc2088a76b635f`
- **SSP Relay:** `5ff4f49e739bfbeafe23514e802e2195e0a1c636`

8.7 HAL-17 - IOS - BIOMETRIC AUTHENTICATION BYPASS

// MEDIUM

Description

The application utilizes biometric authentication mechanisms, including older Touch ID and Face ID technologies, for authentication purposes. These mechanisms are often perceived as robust, leveraging the uniqueness of biometric data like fingerprints. However, vulnerabilities in local authentication implementations can undermine this trust.
Biometric authentication works by validating users locally with stored device credentials, such biometric data (face or fingerprint). This ensures secure and convenient access to app functionality, either by resuming a session with a remote service or performing step-up authentication for sensitive operations.
In this application, a bypass was identified using the Objection framework, which exploits the application's reliance on the native fingerprint evaluation API. Specifically, Objection was used to manipulate the response of the fingerprint reading function (evaluatePolicy) to return a successful authentication result, even when the biometric validation had failed.

Proof of Concept

`Objection` shared overrides the return of the `evaluatePolicy` function so that it returns **True** even if the authentication was unsuccessful.
Video: [Proof of concept]

Score

Impact: 4

Likelihood: 2

Recommendation

1. **Anti-Hook and Anti-Debug Protections**
• Implement runtime checks to detect and prevent hooking and debugging attempts using frameworks like Frida.
• Adopt the IOSSecuritySuite, an open-source solution offering anti-hook, anti-debug, and jailbreak detection functionalities.
Repository: IOS Security Suite
2. **Custom Jailbreak Detection**
• Incorporate additional checks tailored to your application to identify and mitigate tampering or execution on jailbroken devices.
3. **Runtime Code Obfuscation**
• Use runtime obfuscation tools to make reverse engineering and tampering attempts more challenging.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by implementing checks against the identified risk.

8.8 HAL-18 - ANDROID - FINGERPRINT AUTHENTICATION BYPASS

// MEDIUM

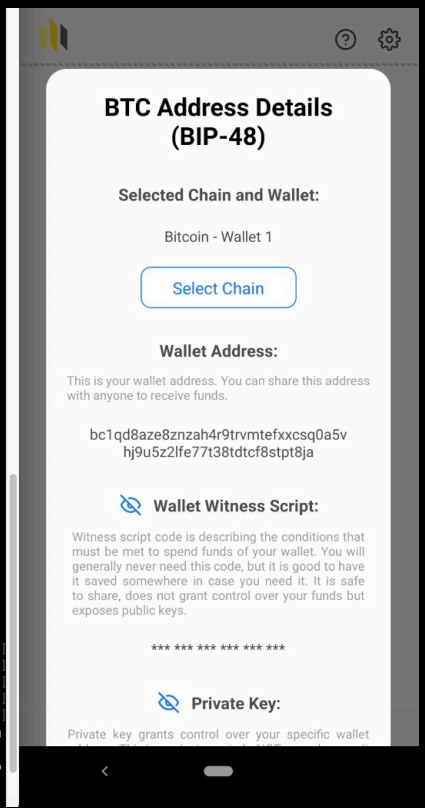
Description

The application employs biometric authentication for critical functionalities, such as accessing sensitive operations like viewing a mnemonic phrase. This implementation, however, is vulnerable to bypass attacks due to improper handling of the BiometricPrompt class. Specifically, the application does not use the CryptoObject parameter during authentication to validate or decrypt critical information securely. The issue lies in the improper reliance on the onAuthenticationSucceeded callback, which can be exploited to simulate a successful authentication. Attackers can hook the `BiometricPrompt.authenticate()` method and manipulate the response, bypassing fingerprint or facial recognition checks without valid biometric data. This vulnerability can expose sensitive application data and compromise its security.
The attack was demonstrated using Frida to hook the `BiometricPrompt.authenticate()` method and force the application to generate a successful authentication response.

Proof of Concept

The attack was demonstrated using Frida to hook the `BiometricPrompt.authenticate()` method and force the application to generate a successful authentication response.

```
ate-android-1.js  -f io.runonflux.sspkey
     ____
    / _  |   Frida 16.5.6 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Pixel 3 (id=89PX0D085)
Spawning `io.runonflux.sspkey`...
Fingerprint hooks loaded!
Spawned `io.runonflux.sspkey`. Resuming main thread!
[Pixel 3::io.runonflux.sspkey ]-> Hooking BiometricPrompt.authenticate()...
Hooking BiometricPrompt.authenticate2()...
Hooking FingerprintManager.authenticate()...
[Cipher.doFinal7()]:   cipherObj: javax.crypto.Cipher@36dbb01
[Cipher.doFinal5()]:   cipherObj: javax.crypto.Cipher@6fe63e7
[Cipher.doFinal5()]:   cipherObj: javax.crypto.Cipher@a435783
[Cipher.doFinal7()]:   cipherObj: javax.crypto.Cipher@df0bff5
[Cipher.doFinal5()]:   cipherObj: javax.crypto.Cipher@9d4cc18
[Cipher.doFinal5()]:   cipherObj: javax.crypto.Cipher@e8e72a9
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@6b29065
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@454ba53
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@80acf90
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@80acf90
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@1387089
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@a47d88e
[Cipher.doFinal7()]:   cipherObj: javax.crypto.Cipher@66a8af
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@c132898
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@816c8f1
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@816c8f1
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@4977b57
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@73c09dc
[Cipher.doFinal5()]:   cipherObj: javax.crypto.Cipher@66a8af
[Pixel 3::io.runonflux.sspkey ]->
[Pixel 3::io.runonflux.sspkey ]->
[Pixel 3::io.runonflux.sspkey ]->
[Pixel 3::io.runonflux.sspkey ]->
[Pixel 3::io.runonflux.sspkey ]->
[Pixel 3::io.runonflux.sspkey ]->
[Pixel 3::io.runonflux.sspkey ]-> [BiometricPrompt.BiometricPrompt()]: cancellationSignal: android.os.CancellationSignal@4f901b2, execu
tor: , callback: androidx.biometric.a$b$a@e362503
cryptoInst:, android.hardware.biometrics.BiometricPrompt$CryptoObject@fa6dc80 class: android.hardware.biometrics.BiometricPrompt$Crypto
Object
[Cipher.doFinal2()]:   cipherObj: javax.crypto.Cipher@5784527
```



Score

Impact: 4

Likelihood: 2

Recommendation

To mitigate this vulnerability, the application should enforce proper usage of the CryptoObject during biometric authentication. Specifically:

1. **Implement Secure Cryptographic Verification**:
Initialize a CryptoObject using a key generated in the Android Keystore with the `setInvalidatedByBiometricEnrollment` property set to **True**. Use this object to encrypt or decrypt a secure key or token.
2. **Require CryptoObject for Authentication**:
Ensure the onAuthenticationSucceeded callback validates the cryptographic operation performed using the CryptoObject. Access to critical data or functions should depend on this validation.
3. **Add Anti-Hooking and Root Detection**:
Implement anti-debugging and anti-hooking mechanisms to prevent attackers from using tools like Frida. Root detection can also help protect the app from exploitation on rooted devices.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by implementing checks against the identified risk.

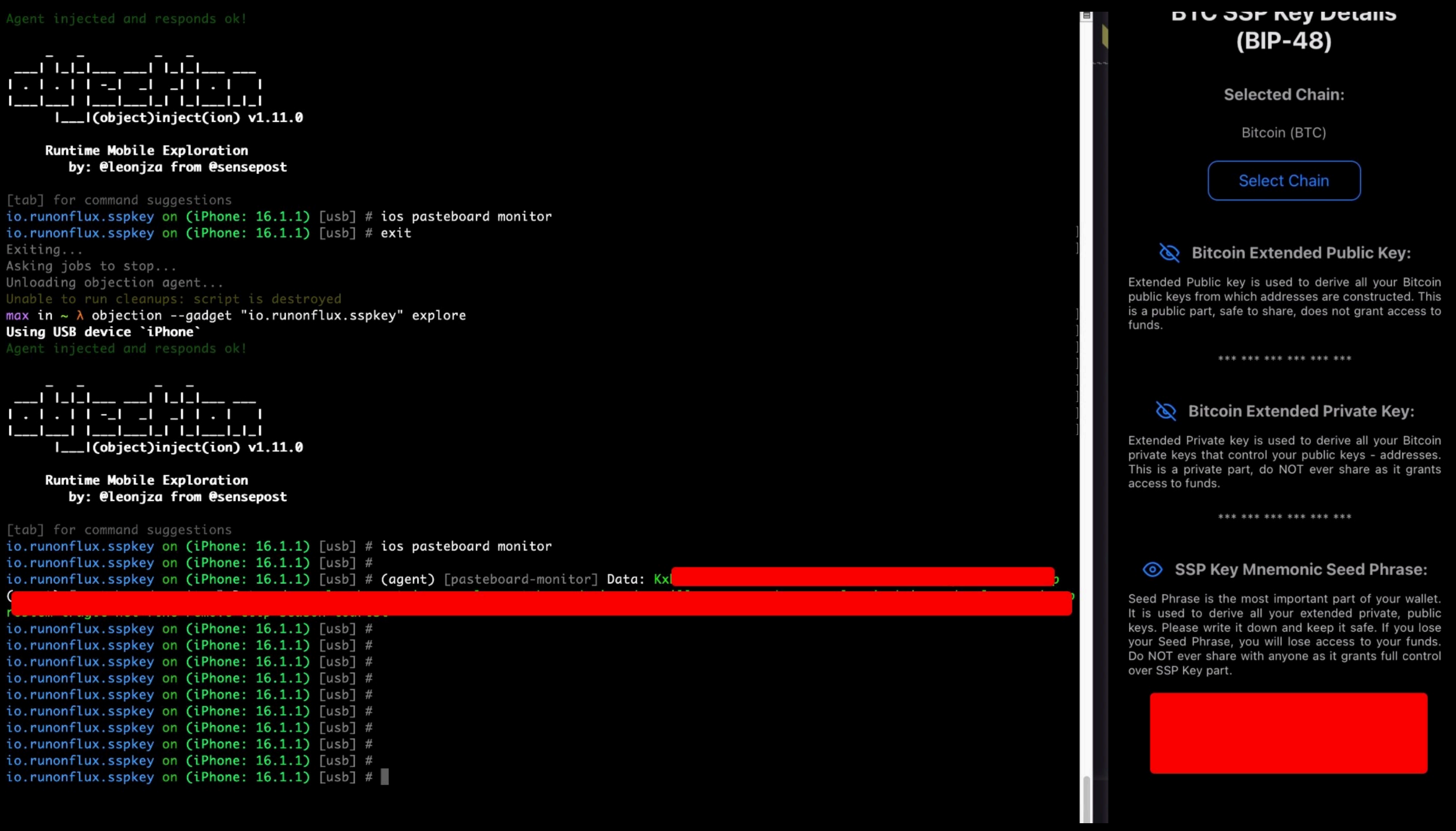
8.9 HAL-31 - IOS - ALLOWING SENSITIVE DATA TO BE COPIED TO CLIPBOARD

// MEDIUM

Description

The SSP Key application allows users to copy sensitive information, such as mnemonics, to the clipboard. While this feature enhances usability, it poses a significant security risk, particularly in environments where clipboard data can be accessed by other devices or applications. For instance, within the Apple ecosystem, clipboard data is often shared across all devices logged into the same iCloud account, such as iPhones, iPads, and Mac computers. This increases the likelihood of unauthorized access to sensitive data if any linked device is compromised or accessed by an unauthorized user. If a mnemonic is intercepted or accessed through clipboard sharing or by malicious applications, attackers could gain control over the wallet, potentially leading to loss of funds and severe privacy breaches.

Proof of Concept



Score

Impact: 4

Likelihood: 2

Recommendation

It is recommended to restrict the ability to copy mnemonics or other sensitive information to the clipboard. Instead, implement alternative, secure methods for sharing sensitive data, encourage users to write down these mnemonics, or implement such as encrypted files for importing/exporting data. If

clipboard copying must be allowed, provide clear warnings to users about the risks associated with copying sensitive information and advise them to clear their clipboard immediately after use.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by splitting the seed phrase and private key into multiple parts during copying, which prevents direct exposure. Additional warnings and explanations have also been added to inform users about the associated risks of copying these sensitive data.

8.10 HAL-07 - ANDROID - INSECURE TRUST OF DEVICE-LEVEL BIOMETRIC AUTHENTICATION

// MEDIUM

Description

The SSP Key Android app implements biometric authentication for user convenience, allowing access to sensitive operations such as viewing mnemonic phrases and signing transactions. However, the app directly trusts the system's biometric settings without requiring re-authentication via PIN after a biometric configuration change (e.g., disabling and re-enabling biometrics or enrolling new biometric profiles). This creates a potential security vulnerability because an attacker with temporary access to an unlocked device can add their biometric data, re-enable biometrics, and subsequently gain unauthorized access to the app without knowing the PIN. The lack of validation when biometrics are reconfigured results in reduced assurance that only the legitimate user can re-enable biometric access after changes to the device's biometric settings.

Proof of Concept

1. Set up the SSP Key app with a PIN and enable biometric authentication.
2. Disable biometric authentication on the device.
3. Re-enable biometrics and enroll a new biometric profile (e.g., a new fingerprint).
4. Open the SSP Key app and use the newly enrolled biometric to authenticate without being prompted for the PIN.
5. Observe that access to sensitive features such as viewing mnemonics is granted without verifying the user's PIN.

Score

Impact: 4

Likelihood: 2

Recommendation

To mitigate this vulnerability, the app should enforce a one-time PIN re-authentication whenever biometric settings are changed on the device. This ensures that the legitimate user is present before granting access to sensitive actions. Additionally, implementing biometric enrollment detection using Android's biometric APIs can enhance security by ensuring that any changes to the enrolled biometrics trigger a PIN re-authentication.

Remediation

SOLVED: The **InFlux Technologies team** solved the issue by implementing checks on changes in the biometrics of the device.

8.11 HAL-08 - IOS - INSECURE TRUST OF DEVICE-LEVEL BIOMETRIC AUTHENTICATION

// MEDIUM

Description

The SSP Key iOS app provides biometric authentication for user convenience, allowing access to sensitive operations such as viewing mnemonic phrases and signing transactions. However, the app directly relies on the system's biometric settings without requiring a PIN re-authentication when biometric configurations are changed (e.g., disabling and re-enabling biometrics or adding new biometric profiles). This creates a significant security risk because an attacker with temporary access to an unlocked device could add their own biometric data, re-enable biometrics, and gain unauthorized access to the app without knowing the original PIN. Since the app does not verify the user's identity through a PIN re-entry after biometric settings are modified, the risk of unauthorized access increases substantially.

Proof of Concept

1. Set up the SSP Key app with a PIN and enable biometric authentication.
2. Disable biometric authentication in the device settings.
3. Re-enable biometrics and add a new biometric profile (e.g., a new Face ID or fingerprint).
4. Open the SSP Key app and use the newly added biometric to authenticate without being prompted for the PIN.
5. Confirm that access to sensitive operations, such as viewing mnemonics, is granted without verifying the user's PIN.

Score

Impact: 4

Likelihood: 2

Recommendation

It is recommended that the app enforces a mandatory PIN re-authentication after any changes to the device's biometric settings are detected. This can be implemented using iOS's LAContext API, which allows biometric enrollment state detection. If any change is detected, sensitive operations should require the user to re-enter their PIN before allowing access.

Remediation

SOLVED: The **InFlux Technologies team** solved the issue by implementing checks on changes in the biometrics of the device.

8.12 HAL-10 - MOBILE - WEAK PASSWORD POLICY

// MEDIUM

Description

The mobile applications permits users to set weak passwords with a minimum length of **4 characters**, which significantly undermines password strength and increases the likelihood of unauthorized access. Additionally, the app does not implement any account lockout or rate limiting mechanism, allowing attackers to attempt unlimited incorrect password entries without being penalized or delayed. This makes the application vulnerable to brute-force attacks, especially in shared environments such as mobile devices and tablets that may be accessed by multiple users.
The risk is further amplified in scenarios where devices are shared among different users or are left unattended without proper security controls. In such cases, an attacker with physical access to the device could exploit the lack of strong passwords and unlimited login attempts to gain unauthorized access to sensitive data stored within the app. Without a proper lockout mechanism, there is no safeguard against repeated guessing of PINs or passwords on shared or compromised devices.

Proof of Concept

```
const setupKey = () => {
  if (password !== passwordConfirm) {
    displayMessage('error', t('cr:err_pins_no_match'));
  } else if (password.length < 4) {
    displayMessage('error', t('cr:err_pins_min_length'));
  } else {
    generateMnemonicPhrase(256);
  }
};
```

Score

Impact: 4

Likelihood: 2

Recommendation

To enhance the security of user accounts and mitigate the risk of brute-force attacks, it is crucial to enforce a minimum password length of 8 characters while requiring a combination of uppercase letters, lowercase letters, numbers, and special characters to ensure robust password complexity. Additionally, implementing an account lockout policy that temporarily disables login after a specified number of consecutive incorrect attempts, such as five failed attempts, can effectively prevent unauthorized access through brute-force methods. To further strengthen security, rate limiting with progressive delays between login attempts should be introduced to deter automated and repeated login attempts by attackers. These measures collectively help to reduce the risk of unauthorized access and enhance overall application security.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by updating the password complexity policy to include a recommendation for stronger passwords. Further, warnings for weak passwords were added, and users were now informed of the risks when selecting weaker passwords.

8.13 HAL-33 - ANDROID - EXPOSURE OF SENSITIVE DATA THROUGH CLIPBOARD

// MEDIUM

Description

The application allows users to copy sensitive data, such as mnemonics or private keys, to the system clipboard. While this feature may provide convenience, it introduces significant security risks. Clipboard data can be accessed by other applications running on the same device, potentially allowing unauthorized access to sensitive information. This issue becomes even more critical in multi-device environments where clipboard contents can be shared across devices.
Copying mnemonics or private keys to the clipboard creates a risk of data leakage and unauthorized access, potentially leading to the loss of funds or compromise of user accounts. In Android environments, clipboard data is often accessible by other applications without requiring explicit user permissions, increasing the risk of exposure.

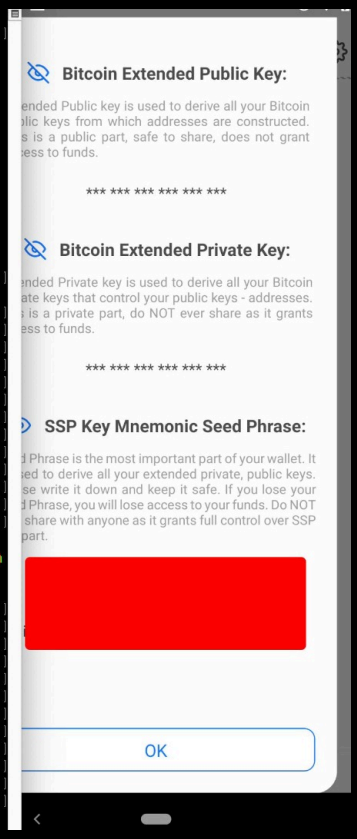
Proof of Concept

```
-  exploit                    com.example.exploit
max in ~ λ objection --gadget "io.runonflux.sspkey" explore
Using USB device `Pixel 3`
Agent injected and responds ok!

     _     _       _     _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | |_ -|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.11.0

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
io.runonflux.sspkey on (google: 11) [usb] # android clipboard monitor
(agent) Warning! This module is still broken. A pull request fixing it would be awesome!
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] # (agent) [pasteboard-monitor] Data: Kwh[redacted]
(agent) [pasteboard-monitor] Data: [redacted]
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
```



Score

Impact: 4

Likelihood: 2

Recommendation

To mitigate this risk, the application should avoid copying sensitive data, such as mnemonics and private keys, to the clipboard. Instead, implement more secure alternatives, such as:

• Providing a secure in-app mechanism to temporarily view sensitive data without copying.
• Allowing users to input mnemonics or private keys directly without exposing them to the clipboard.
• If clipboard functionality is absolutely necessary, clear clipboard contents immediately after use or limit the ability to copy sensitive information only to trusted applications.
• Implement secure warnings to inform users of the risks associated with copying sensitive data.
Instead, implement alternative, secure methods for sharing sensitive data, such as encrypted files for importing/exporting data. By avoiding the use of the clipboard for sensitive data or implementing stringent controls, the risk of unauthorized access can be significantly reduced.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by splitting the seed phrase and private key into multiple parts during copying, which prevents direct exposure. Additional warnings and explanations have also been added to inform users about the associated risks of copying these sensitive data.

8.14 HAL-25 - BE - POTENTIAL RISK DUE TO THIRD-PARTY IFRAME

// MEDIUM

Description

The wallet browser extension integrates a third-party iframe to facilitate cryptocurrency purchases. However, the iframe is granted extensive permissions, including access to scripts, same-origin content, popups, and sensitive APIs such as the camera, microphone, gyroscope, accelerometer, payment, and autoplay. Although these permissions are necessary for proper functionality as confirmed by the third-party service provider, they significantly expand the attack surface. If the third-party provider is compromised or malicious scripts are injected, it could lead to phishing attacks, user interface (UI) redressing, or unauthorized data capture. Without adequate control or monitoring of the iframe, the extension remains vulnerable to these risks. Additionally, the current Content Security Policy (CSP) configuration lacks frame-source restrictions, which could allow unauthorized iframes to be embedded, further increasing the risk of content injection and malicious activity.

Score

Impact: 4

Likelihood: 2

Recommendation

It is recommended to strengthen the Content Security Policy (CSP) by explicitly limiting frame sources to trusted origins only. For example, use `"frame-src 'self' https://buy.onramper.com"` to ensure that only the intended iframe from a trusted origin is allowed, preventing rogue or unauthorized iframes from being loaded.
Additionally, consider implementing a kill switch mechanism that allows you to remotely disable the iframe functionality in case the third-party service is compromised. This can be achieved by checking a remote configuration file or flag before loading the iframe.
User awareness is also crucial in mitigating potential risks. Adding a clear warning to users about the nature of third-party integrations and encouraging them to trust the provider before proceeding will improve transparency and reduce the risk of phishing or malicious interaction.

Remediation

SOLVED: The **InFlux Technologies team** resolved the issue by implementing the appropriate checks.

Remediation Hash

https://github.com/RunOnFlux/ssp-wallet/pull/376

References

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-src

8.15 HAL-30 - API - OUTDATED VERSIONS OF TLS SUPPORTED

// LOW

Description

The relay server relay.ssp.runonflux.io supported outdated and deprecated versions of the TLS protocol, specifically TLS 1.0 and TLS 1.1. These versions are known to have vulnerabilities and are no longer recommended for secure communication. Additionally, modern browsers and client applications have discontinued support for these protocols due to their weak security features, potentially exposing the server and clients to attacks like protocol downgrade or MITM (Man-In-The-Middle).
These outdated TLS versions have been superseded by newer versions (e.g., TLS v1.3) which provide enhanced security features and address many of the vulnerabilities found in their predecessors. By supporting these older versions, the application not only endangers the integrity and confidentiality of data in transit but also risks non-compliance with modern security standards and regulations.

Proof of Concept

```
Testing all IPv4 addresses (port 443): 104.18.6.165 104.18.7.165
---------------------------------------------------------------------------------------------------------------------
 Start 2025-01-16 15:40:27        -->> 104.18.6.165:443 (relay.ssp.runonflux.io) <<--

 Further IP addresses:   104.18.7.165 2606:4700::6812:6a5 2606:4700::6812:7a5
 rDNS (104.18.6.165):    --
 Service detected:       HTTP


 Testing protocols via sockets except NPN+ALPN

 SSLv2      not offered (OK)
 SSLv3      not offered (OK)
 TLS 1      offered (deprecated)
 TLS 1.1    offered (deprecated)
 TLS 1.2    offered (OK)
 TLS 1.3    offered (OK): final
 NPN/SPDY   not offered
 ALPN/HTTP2 h2, http/1.1 (offered)

 Testing cipher categories

 NULL ciphers (no encryption)                not offered (OK)
 Anonymous NULL Ciphers (no authentication)  not offered (OK)
 Export ciphers (w/o ADH+NULL)               not offered (OK)
 LOW: 64 Bit + DES, RC[2,4] (w/o export)     not offered (OK)
 Triple DES Ciphers / IDEA                   offered
 Obsolete CBC ciphers (AES, ARIA etc.)       offered
 Strong encryption (AEAD ciphers)            offered (OK)


 Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption, 3DES, RC4

 PFS is offered (OK)          TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 ECDHE-RSA-AES256-GCM-SHA384 ECDHE-ECDSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-SHA384
                              ECDHE-ECDSA-AES256-SHA384 ECDHE-RSA-AES256-SHA ECDHE-ECDSA-AES256-SHA ECDHE-ECDSA-CHACHA20-POLY1305 ECDHE-RSA-CHACHA20-POLY1305 TLS_AES_128_GCM_SHA256
                              ECDHE-RSA-AES128-GCM-SHA256 ECDHE-ECDSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-ECDSA-AES128-SHA256 ECDHE-RSA-AES128-SHA ECDHE-ECDSA-AES128-SHA
 Elliptic curves offered:     prime256v1 secp384r1 secp521r1 X25519


 Testing server preferences

 Has server cipher order?     yes (OK) -- only for < TLS 1.3
 Negotiated protocol          TLSv1.3
 Negotiated cipher            TLS_AES_256_GCM_SHA384, 253 bit ECDH (X25519)
 Cipher order
    TLSv1:     ECDHE-RSA-AES128-SHA AES128-SHA ECDHE-RSA-AES256-SHA AES256-SHA DES-CBC3-SHA
```

Score

Impact: 2

Likelihood: 2

Recommendation

It is recommended to review and update the TLS configuration across all servers according to the following best practices:
- Disable support for outdated protocols such as TLS 1.0 and TLS 1.1.

- Remove support for weak cipher suites.
- Use strong encryption methods, ensuring key lengths of at least 256 bits for symmetric encryption and 4096 bits for RSA. For ECC algorithms, use a minimum key size of 512 bits.

Remediation

SOLVED: The **InFlux Technologies team** resolved the issue by disabling the support for outdated TLS versions.

References

https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Security_Cheat_Sheet.html

8.16 HAL-14 - API - CACHEABLE HTTPS RESPONSE

// LOW

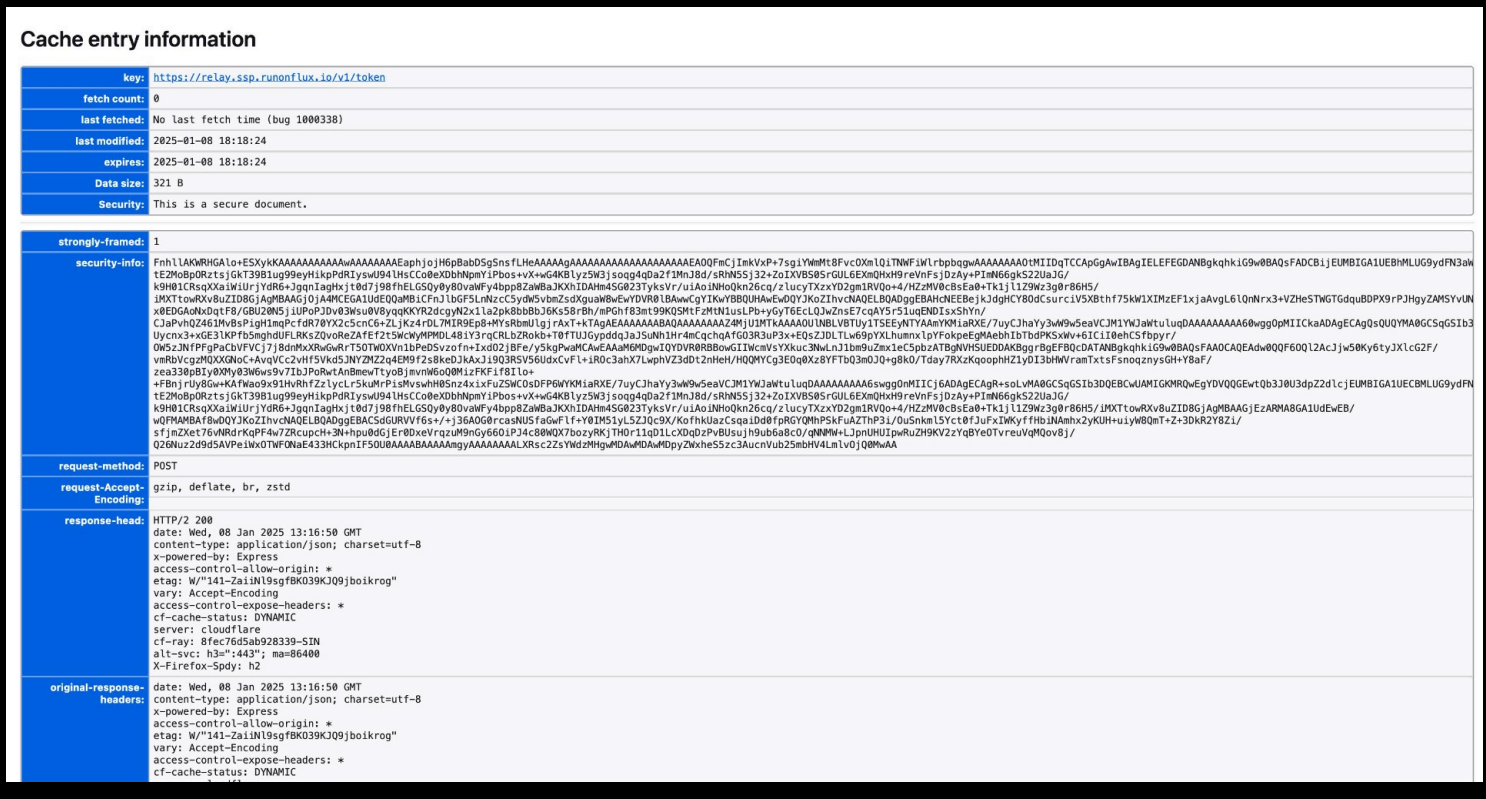
Description

The API service is currently returning cacheable HTTPS responses on certain endpoints. This behavior is risky because sensitive information sent over HTTPS can be cached on the client side, potentially exposing it to unauthorized users who have access to the client's device.
Caching behavior observed indicates that HTTP response headers from the API service do not adequately prevent caching, missing headers like:

- `Cache-Control: no-store, no-cache, must-revalidate, private`
- `Pragma: no-cache`
- `Expires: 0`

The absence of these headers allows client-side caching of potentially sensitive information, leading to risks such as data breaches and unauthorized access.

Proof of Concept



Cache entry information

key:	https://relay.ssp.runonflux.io/v1/token
fetch count:	0
last fetched:	No last fetch time (bug 1000338)
last modified:	2025-01-08 18:18:24
expires:	2025-01-08 18:18:24
Data size:	321 B
Security:	This is a secure document.
strongly-framed:	1
security-info:	FnhllAKWRHGAlo+ESXykKAAAAAAAAAAwAAAAAAAAEaphjojH6pBabDSgSnsfLHeAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAEAOQFmCjImkVxP+7sgiYWmMt8FvcOXmlQiTNWFiWlrbpbqgwAAAAAAAAOtMIIDqTCCApGgAwIBAgIELEFEGDANBgkqhkiG9w0BAQsFADCBijEUMBIGA1UEBhMLUG9ydFN3aWtE2MoBpORztsjGkT39B1ug99eyHikpPdRIyswU94lHsCCo0eXDbhNpmYiPbos+vX+wG4KBlyz5W3jsoqg4qDa2f1MnJ8d/sRhN55j32+ZoIXVBS0SrGUL6EXmQHxH9reVnFsjDzAy+PImN66gkS22UaJG/ k9H01CRsqXXaiWiUrjYdR6+JgqnIagHxjt0d7j98fhELGSQy0y8OvaWFy4bpp8ZaWBaJKXhIDAHm4SG023TyksVr/uiAoiNHoQkn26cq/zlucyTXzxYD2gm1RVQo+4/HZzMV0cBsEa0+Tk1jlZ9Wz3g0r86H5/ iMXTtowRXv8uZID8GjAgMBAAGjOjA4MCEGA1UdEQQaMBiCFnJlbGF5LnNzcC5ydW5vbmZsdXguaW8wEwYDVR0lBAwwCgYIKwYBBQUHAwEwDQYJKoZIhvcNAQELBQADggEBAHcNEEBejkJdgHCY8OdCsurciV5XBthf75kW1XIMzEF1xjaAvgL6lQnNrx3+VZHeSTWGTGdquBDPX9rPJHgyZAMSYvUNx0EDGAoNxDqtF8/GBU20N5jiUPoPJDv03Wsu0V8yqqKKYR2dcgyN2x1la2pk8bbBbJ6Ks58rBh/mPGhf83mt99KQSMtFzMtN1usLPb+yGyT6EcLQJwZnsE7cqAY5r51uqENDIsxShYn/ CJaPvhQZ46lMvBsPigH1mqPcfdR70YX2c5cnC6+ZLjKz4rDL7MIR9Ep8+MYsRbmUlgjrAxT+kTAgAEAAAAAABAQAAAAAAAZ4MjU1MTkAAAAOUlNBLVBTUy1TSEEyNTYAAmYkMiaRXE/7uyCJhaYy3wW9w5eaVCJM1YWJaWtuluqDAAAAAAAA60wggOpMIICkaADAgECAgQsQUQYMA0GCSqGSIb3 Uycnx3+xGE3lKPfb5mghdUFLRKsZQvoReZAfEf2t5WckWyMPMDL48iY3rqCRLbZRokb+T0TfUJGypddqJaJSuNh1Hr4mCqchqAfGQ3R3uP3x+EQsZJDLTLw69pYXLhumrxlpYFokpeEgMAebhIbTbdPKSxWv+6ICiI0ehCSfbpyr/ OW5ZJNfPFgPaCbVFVCj7j8dnMxXRwGwRrT5OTWDXVn1bPeDSvzofn+IxdO2jBFe/y5kgPwaMCAwEAAaMGMDgwIQYDVR0RBBowGIIWcmVsYXkuc3NwLnJlbm9uZmx1eC5pbzATBgNVHSUEDDAKBggrBgEFBQcDATANBgkqhkiG9w0BAQsFAAOCAQEAdw0QQF6OQl2AcJjw50Ky6tyJXlcG2F/ vmRbVcgzMQXXGNoC+AvqVCc2vHf5Vkd5JNYZMZZq4EM9f2s8keDJkAxJi9Q3RSV56UdxCvFl+iROc3ahX7LwphVZ3dDt2nHeH/HQQMYCg3EOq0Xz8YFTbQ3mOJQ+g8kO/Tday7RXzKqoophHZ1yDI3bHWVramTxtsFsnoqznysGH+Y8aF/ zea330pBIy0XMy03W6ws9v7IbJPoRwtAnBmewTtyoBjmvnW6oQ0MizFKFif8Ilo+ +FBnjrUy8Gw+KAfWao9x91HvRhfZzlycLr5kuMrPisMvswhH0Snz4xixFuZSWCOsDFP6WYKMiaRXE/7uyCJhaYy3wW9w5eaVCJM1YWJaWtuluqDAAAAAAAA6swggOnMIICj6ADAgECAgR+soLvMA0GCSqGSIb3DQEBCwUAMIGKMRQwEgYDVQQGEwtQb3J0U3dpZ2dlcjEUMBIGA1UECBMLUG9ydFN tE2MoBpORztsjGkT39B1ug99eyHikpPdRIyswU94lHsCCo0eXDbhNpmYiPbos+vX+wG4KBlyz5W3jsoqg4qDa2f1MnJ8d/sRhN55j32+ZoIXVBS0SrGUL6EXmQHxH9reVnFsjDzAy+PImN66gkS22UaJG/ k9H01CRsqXXaiWiUrjYdR6+JgqnIagHxjt0d7j98fhELGSQy0y8OvaWFy4bpp8ZaWBaJKXhIDAHm4SG023TyksVr/uiAoiNHoQkn26cq/zlucyTXzxYD2gm1RVQo+4/HZzMV0cBsEa0+Tk1jlZ9Wz3g0r86H5/iMXTtowRXv8uZID8GjAgMBAAGjEzARMA8GA1UdEwEB/ wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBACSdGURVVf6s+/+j36AOG0rcasNU5faGwFlf+Y0IM51yL5ZJQc9X/KofhkUazCsqaiDd0fpRGYQMhPSkFuAZThP3i/OuSnkml5Yct0fJuFxIWKyffHbiNAmhx2yKUH+uiyW8QmT+Z+3DkR2Y8Zi/ sfjmZXet76vNRdrKqPF4w7ZRcupcH+3N+hpu0dGjEr0DxeVrqzuM9nGy6GOiPJ4c80WQX7bozyRKjTHOr11qD1LcXDqDzPvBUsujh9ub6a8cO/qNNMW+LJpnUHUIpwRuZH9KV2zYqBYeOTvreuVqMQov8j/ Q26Nuz2d9d5AVPeiWxOTWFONaE433HCkpnIF5OU0AAABAAAAAmgyAAAAAAAALXRsc2ZsYWdzMHgwMDAwMDAwMDpyZWxheS5zc3AucnVub25mbHV4LmlvOjQ0MwAA	
request-method:	POST
request-Accept-Encoding:	gzip, deflate, br, zstd
response-head:	HTTP/2 200 date: Wed, 08 Jan 2025 13:16:50 GMT content-type: application/json; charset=utf-8 x-powered-by: Express access-control-allow-origin: * etag: W/"141-ZaiiNl9sgfBKO39KJQ9jboikrog" vary: Accept-Encoding access-control-expose-headers: * cf-cache-status: DYNAMIC server: cloudflare cf-ray: 8fec76d5ab928339-SIN alt-svc: h3=":443"; ma=86400 X-Firefox-Spdy: h2
original-response-headers:	date: Wed, 08 Jan 2025 13:16:50 GMT content-type: application/json; charset=utf-8 x-powered-by: Express access-control-allow-origin: * etag: W/"141-ZaiiNl9sgfBKO39KJQ9jboikrog" vary: Accept-Encoding access-control-expose-headers: * cf-cache-status: DYNAMIC

```
00000000:  7b  22  73  74  61  74  75  73  22  3a  22  73  75  63  63  65  {"status":"succe
00000010:  73  73  22  2c  22  64  61  74  61  22  3a  7b  22  6b  65  79  ss","data":{"key
00000020:  54  6f  6b  65  6e  22  3a  22  65  45  6f  37  6a  51  76  31  Token":"eEo7jQv1
00000030:  44  45  50  71  6c  30  4e  42  46  78  41  65  61  4f  3a  41  DEPql0NBFxAeaO:A
00000040:  50  41  39  31  62  45  2d  6d  49  66  67  76  79  55  75  62  PA91bE-mIfgvyUub
00000050:  56  41  59  4c  37  59  4d  57  4e  56  6b  76  42  44  52  77  VAYL7YMWNVkvBDRw
00000060:  70  75  6a  42  4a  59  74  34  32  7a  66  6e  52  55  71  39  pujBJYt42zfnRUq9
00000070:  4b  4d  57  74  31  46  67  73  39  49  67  6d  4c  42  66  42  KMWt1Fgs9IgmLBfB
00000080:  73  42  47  48  6b  50  43  38  34  4a  56  37  64  36  35  6f  sBGHkPC84JV7d65o
00000090:  5f  41  52  35  44  47  7a  77  67  79  7a  44  57  54  72  61  _AR5DGzwgyzDWTra
000000a0:  6d  47  50  59  64  66  58  5f  34  61  64  42  50  53  62  44  mGPYdfX_4adBPSbD
000000b0:  43  4b  57  33  2d  6b  22  2c  22  63  72  65  61  74  65  64  CKW3-k","created
000000c0:  41  74  22  3a  22  32  30  32  34  2d  31  32  2d  33  31  54  At":"2024-12-31T
000000d0:  32  30  3a  35  30  3a  33  38  2e  31  35  34  5a  22  2c  22  20:50:38.154Z","
000000e0:  77  61  6c  6c  65  74  54  6f  6b  65  6e  22  3a  6e  75  6c  walletToken":nul
000000f0:  6c  2c  22  77  6b  49  64  65  6e  74  69  74  79  22  3a  22  l,"wkIdentity":"
00000100:  62  63  31  71  77  37  37  76  35  77  77  34  64  61  34  70  bc1qw77v5ww4da4p
00000110:  34  38  79  7a  30  33  37  32  37  65  6b  6a  79  65  6d  79  48yz03727ekjyemy
00000120:  33  70  65  72  6a  6e  6a  74  75  6d  37  66  38  6c  75  75  3perjnjtum7f8luu
00000130:  68  66  76  6e  77  67  32  71  79  66  6b  74  36  64  22  7d  hfvnwg2qyfkt6d"}
00000140:  7d                                                             }
```

Score

Impact: 2

Likelihood: 2

Recommendation

It is advised to include the necessary HTTP headers on all sensitive endpoints to prevent caching of any sensitive information or responses that should not be stored on the client side. Specifically:
- Set `Cache-Control` to "`no-store, no-cache, must-revalidate, private`"
- Add `Pragma: no-cache`
- Use `Expires: 0`
These headers will instruct the client's browser not to store the sensitive content, reducing the risk of data exposure.

Remediation

SOLVED: The **InFlux Technologies team** resolved the issue by implementing the appropriate headers as per the recommendations.

8.17 HAL-15 - IOS - LACK OF JAILBREAK DETECTION MECHANISM

// LOW

Description

The iOS application lacks anti-jailbreak mechanisms, exposing it to potential exploitation by adversaries. Jailbreaking allows attackers to bypass the inherent restrictions of the iOS operating system, enabling actions such as reverse engineering, application modification, and unauthorized use of modified app versions. While no anti-jailbreak solution is entirely foolproof against a determined adversary, such mechanisms significantly increase the effort required to exploit an application and serve as a critical component of a defense-in-depth strategy.

Score

Impact: 2

Likelihood: 2

Recommendation

1. **Add Open-Source Jailbreak Detection Libraries**
• Integrate tools such as IOSSecuritySuite or other frameworks to detect common jailbreak indicators like unauthorized files, processes, or libraries.
2. **Custom Jailbreak Detection**
• Implement tailored checks to identify specific jailbreak artifacts or behaviors, such as:
• Existence of known jailbreak files (e.g., Cydia.app, MobileSubstrate).
• Writable access to system directories (e.g., /private/var).
• Presence of dynamic libraries loaded into the app.
3. **Frida Detection**
• Add runtime detection for debugging and hooking tools such as Frida to prevent runtime tampering or reverse engineering.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by detecting rooted/jailbroken devices, notifying users about the security risks while ensuring minimal impact on user experience.

8.18 HAL-16 - ANDROID - LACK OF ROOT DETECTION MECHANISM

// LOW

Description

Anti-root mechanisms was 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.

Score

Impact: 2

Likelihood: 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.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by detecting rooted/jailbroken devices, notifying users about the security risks while ensuring minimal impact on user experience.

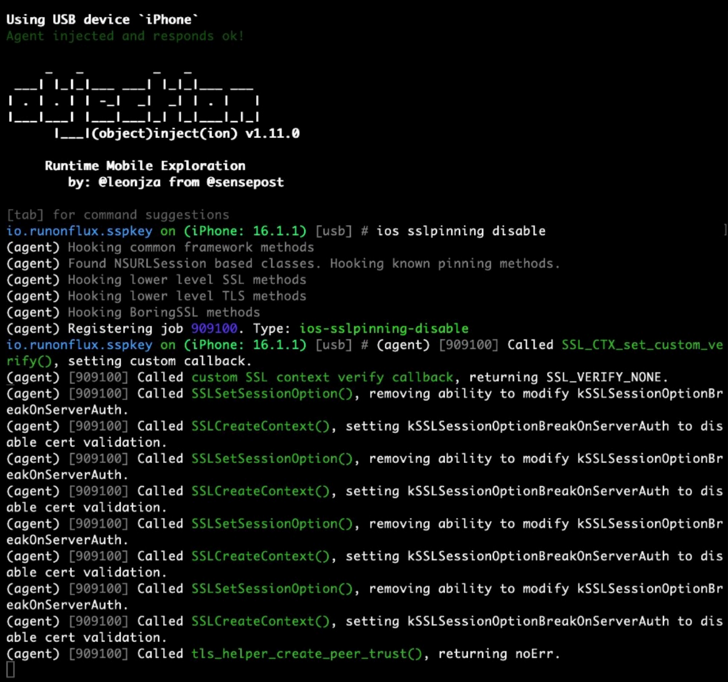
8.19 HAL-12 - IOS - CERTIFICATE PINNING BYPASS

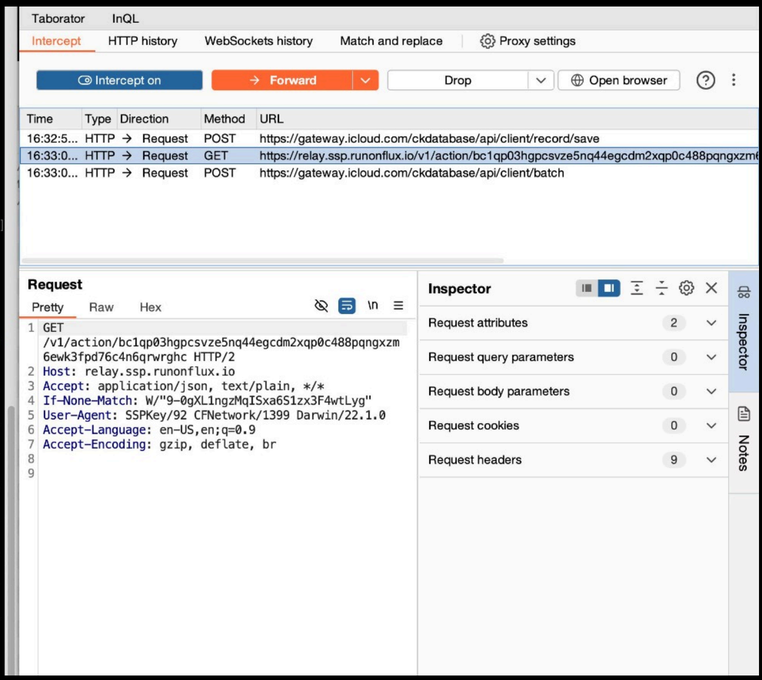
// LOW

Description

Certificate pinning is a security practice that binds an application to a specific X.509 certificate or public key, ensuring that it only connects to the designated server. By bypassing the need to trust external Certificate Authorities (CAs), this technique helps reduce exposure to man-in-the-middle (MitM) attacks. However, if not properly implemented, attackers can intercept communications by bypassing SSL pinning, exposing sensitive data like user credentials and session IDs.
In the case of target, although SSL pinning is implemented, it is vulnerable due to the use of methods with predictable names and a lack of anti-hooking mechanisms. This allows attackers to bypass SSL pinning, potentially leading to unauthorized interception of sensitive data and communication between the application and its backend services.

Proof of Concept





Score

Impact: 2

Likelihood: 2

Recommendation

To mitigate this vulnerability, implement jailbreak/root detection by checking for known files, directories (e.g., Cydia, su, Magisk), and rooting services. Integrate anti-hooking and anti-tampering mechanisms to strengthen SSL pinning and prevent data interception. Additionally, obfuscate the SSL pinning logic using tools like ProGuard or DexGuard to hinder reverse engineering and increase security robustness.

Remediation

FUTURE RELEASE: The **InFlux Technologies team** will address the issue in the future builds of the applications.

8.20 HAL-13 - ANDROID - CERTIFICATE PINNING BYPASS

// LOW

Description

Certificate pinning is a security measure that binds a mobile app to a specific X.509 certificate or public key for backend communication, rather than relying on certificates from any trusted certificate authority (CA). This reduces the risk of man-in-the-middle (MitM) attacks that exploit compromised or forged CAs by ensuring the app only trusts a pre-defined certificate.
In this case, the target application has implemented SSL pinning to verify server authenticity during network communications. However, it uses methods with commonly identifiable names and lacks anti-hooking mechanisms, allowing attackers to bypass certificate pinning through runtime modifications. This vulnerability enables attackers to intercept sensitive information, such as authentication tokens, by circumventing the app's SSL pinning protection. Although certificate pinning is an effective defense mechanism, without proper tamper resistance, it becomes vulnerable to manipulation and may not prevent determined attackers from intercepting HTTP traffic.

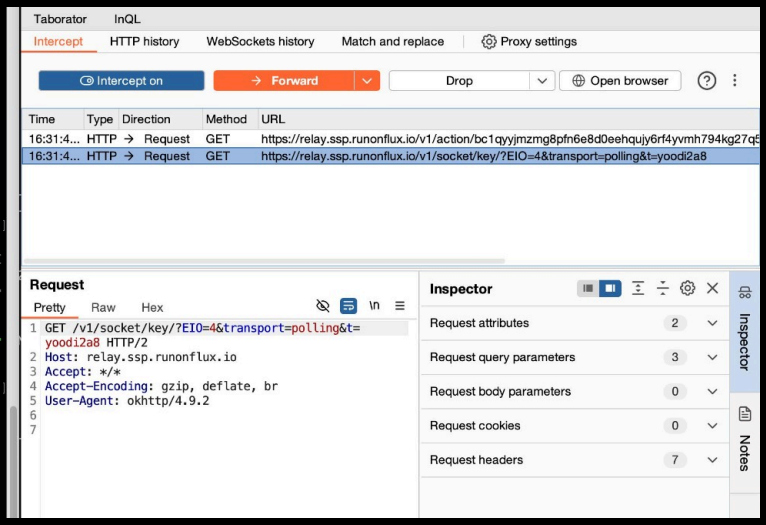
Proof of Concept

```
Using USB device `Pixel 3`
Agent injected and responds ok!

     _ _      _ _
  ___| |_|_|___ ___| |_|_|___ ___
 | . | | | |   |_ -|  _| | . |   |
 |___|___| |_|_|___|_| |_|___|_|_|
       |___|(object)inject(ion) v1.11.0

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
[io.runonflux.sspkey on (google: 11) [usb] # android sslpinning disable
(agent) Custom TrustManager ready, overriding SSLContext.init()
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustManagerImpl.verifyC
hain()
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustManagerImpl.checkTr
ustedRecursive()
(agent) Registering job 179318. Type: android-sslpinning-disable
io.runonflux.sspkey on (google: 11) [usb] # (agent) [179318] Called (Android 7+) TrustManager
Impl.checkTrustedRecursive(), not throwing an exception.
io.runonflux.sspkey on (google: 11) [usb] #
[io.runonflux.sspkey on (google: 11) [usb] #
io.runonflux.sspkey on (google: 11) [usb] #
```



Score

Impact: 2

Likelihood: 2

Recommendation

To mitigate this vulnerability, implement jailbreak/root detection by checking for known files, directories (e.g., Cydia, su, Magisk), and rooting services. Integrate anti-hooking and anti-tampering mechanisms to strengthen SSL pinning and prevent data interception. Additionally, obfuscate the SSL pinning logic using tools like ProGuard or DexGuard to hinder reverse engineering and increase security robustness.

Remediation

FUTURE RELEASE: The **InFlux Technologies team** will address the issue in the future builds of the applications.

8.21 HAL-35 - IOS - LACK OF ANTI-TAMPERING AND ANTI-HOOKING MECHANISMS

// LOW

Description

The application lack anti-tampering and anti-hooking mechanisms to detect and mitigate unauthorized modifications, runtime manipulations, and hooking attempts. These features enhance application security by ensuring file integrity, blocking execution in untrusted environments, and identifying tools like Frida or dynamic frameworks. Properly implemented, these mechanisms protect sensitive components and prevent data exposure.

Score

Impact: 2

Likelihood: 2

Recommendation

To enhance the security of the application, implement cryptographic checksum validation to ensure the integrity of application files and resources at runtime. Enforce signature verification to detect unauthorized modifications or re-signing of the app. Integrate runtime integrity checks to monitor for debugging, hooking tools, or unauthorized tampering during the app's execution. For iOS, implement jailbreak detection to identify unauthorized environments.

Remediation

FUTURE RELEASE: The **InFlux Technologies team** will address the issue in the future builds of the applications.

8.22 HAL-34 - ANDROID - LACK OF ANTI-TAMPERING AND ANTI-HOOKING MECHANISMS

// LOW

Description

The application lack anti-tampering mechanisms, emulator detection, and anti-hooking features to protect against runtime manipulation, reverse engineering, and unauthorized modifications. These measures enhance the app's resilience by detecting unauthorized file changes, blocking execution in emulator environments, and mitigating the use of hooking tools like Frida. Proper implementation of these features ensures the security of sensitive application components and data.

Score

Impact: 2

Likelihood: 2

Recommendation

To enhance the security of the application, implement cryptographic checksum validation to ensure the integrity of application files and resources at runtime. Enforce signature verification to detect unauthorized modifications or re-signing of the app. Integrate runtime integrity checks to monitor for debugging, hooking tools, or unauthorized tampering during the app's execution. Android, utilize root detection mechanisms to restrict functionality on rooted devices.

Remediation

FUTURE RELEASE: The **InFlux Technologies team** will address the issue in the future builds of the applications.

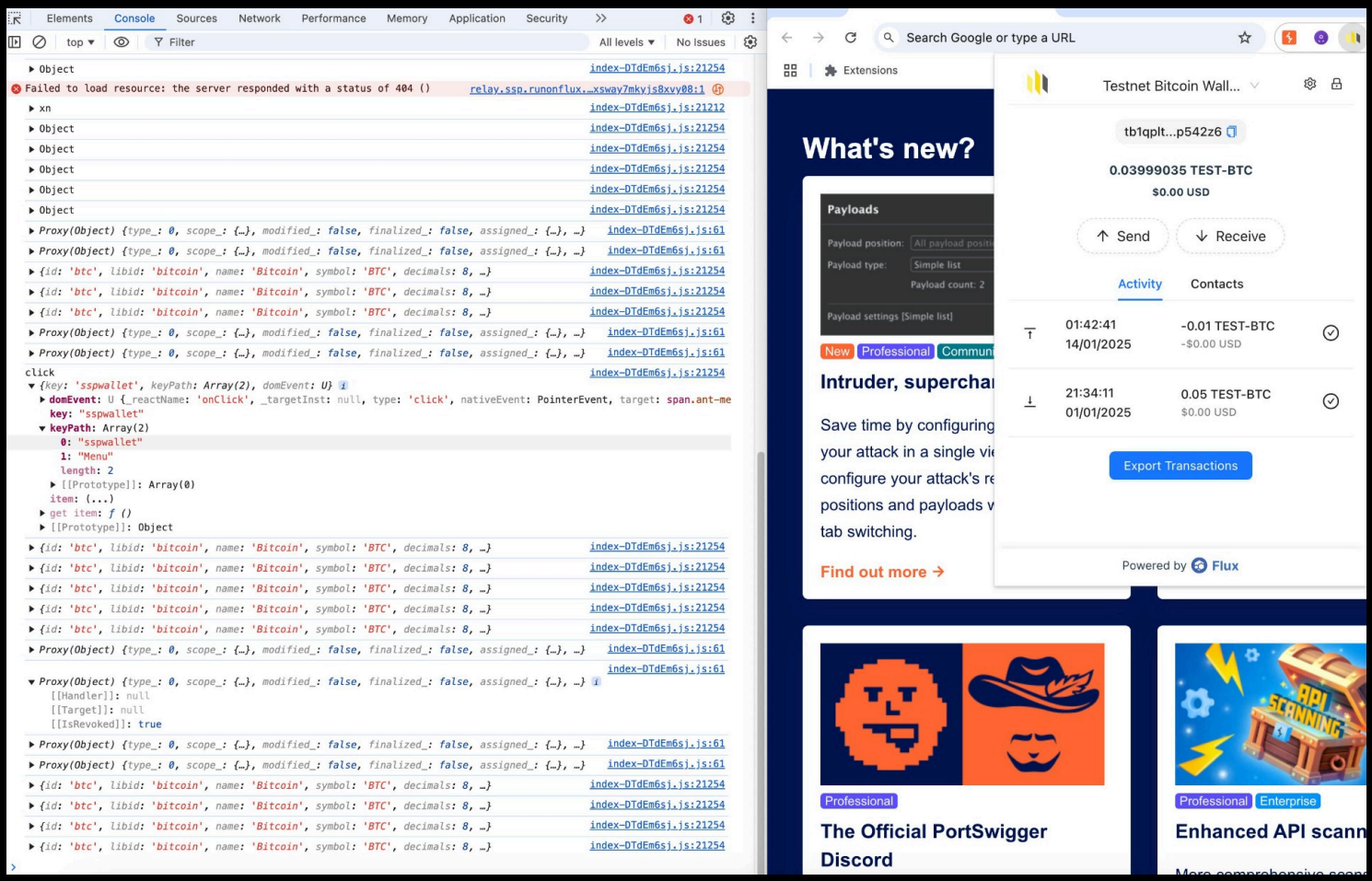
8.23 HAL-28 - BE - VERBOSE LOGGING IN EXTENSION

// LOW

Description

The browser extension had verbose logging enabled, which outputs information such as internal application data, actions, and key interactions to the browser console. While logging can be helpful during development, leaving verbose logging enabled in production poses a security risk, as it may expose sensitive details such as wallet IDs, transaction data, and other internal logics. Verbose logs can potentially be exploited by malicious actors who have physical or remote access to a user's device. They can gain insights into the internal workings of the application, user interactions, and potentially sensitive data, which can lead to targeted attacks, data theft, or unauthorized access.

Proof of Concept



Score

Impact: 2

Likelihood: 2

Recommendation

It is recommended to disable verbose logging in the production builds to prevent any data from being exposed. If logging is necessary for debugging or monitoring purposes, ensure that:

• Only minimal and non-sensitive information is logged.

• Debug logs are conditionally enabled only in development environments.

Remediation

SOLVED: The **InFlux Technologies team** resolved the issue by disabling the logs for production builds.

Remediation Hash

https://github.com/RunOnFlux/ssp-wallet/pull/376

8.24 HAL-06 - BE - LACK OF PASSWORD COMPLEXITY AND PASSWORD POLICY

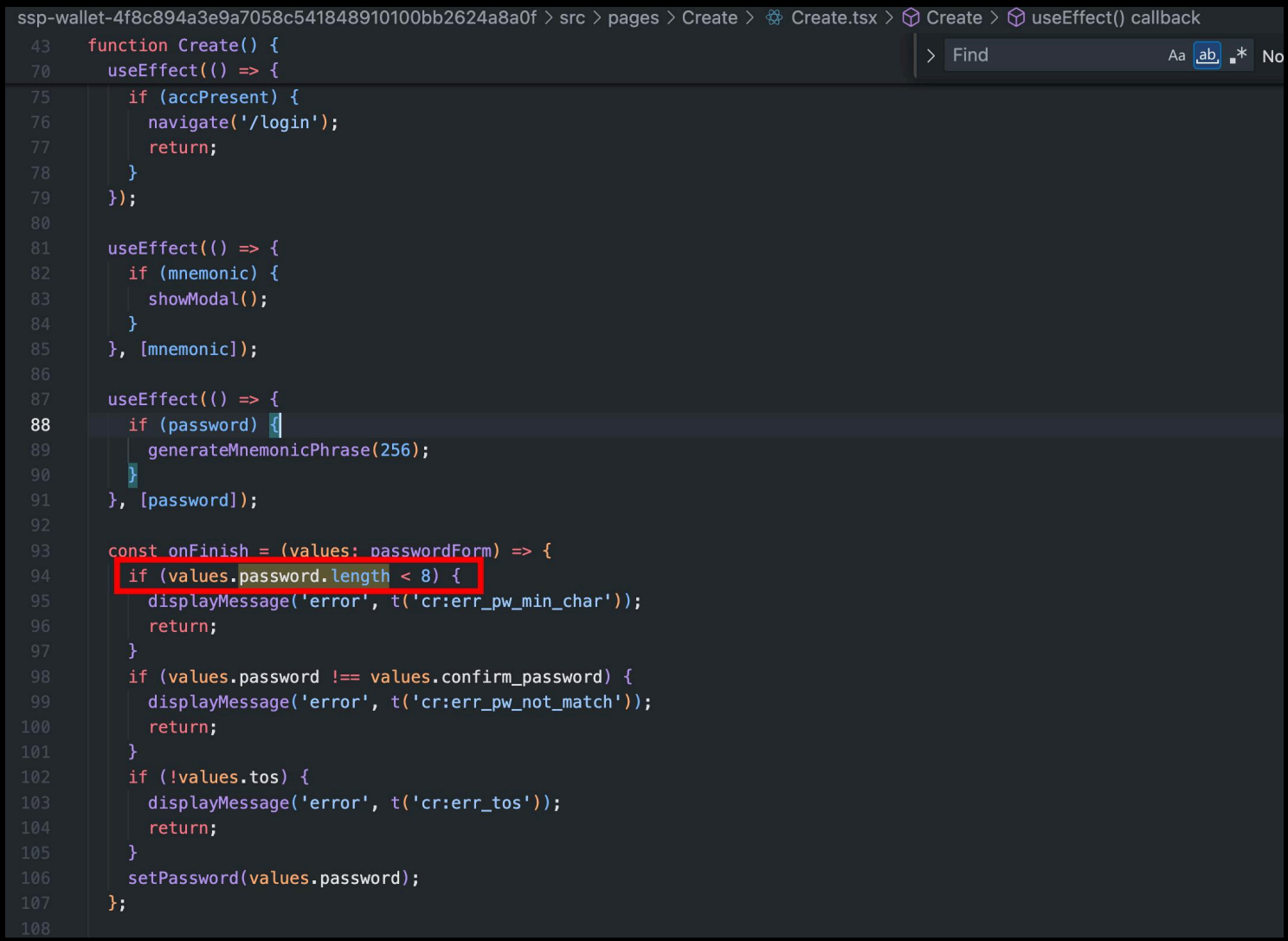
// LOW

Description

The browser extension enforces only a minimal password length of 8 characters without requiring additional complexity, such as a combination of uppercase and lowercase letters, numbers, or special characters. Additionally, it lacks a robust password policy with essential security measures, such as checking for commonly used or weak passwords, or lacks a lockout mechanism. This inadequate password enforcement increases the risk of unauthorized access to sensitive extension data, including encrypted private keys and seed phrases, as simple passwords are more susceptible to brute-force or guessing attacks.

Proof of Concept

```
if (values.password.length < 8) {
  displayMessage('error', t('cr:err_pw_min_char'));
  return;
}
```



Score

Impact: 2

Likelihood: 2

Recommendation

It is recommended to implement a stronger password policy that enforces greater complexity requirements, such as a mix of uppercase and lowercase letters, numbers, and special characters. Additionally, the extension should incorporate mechanisms to check for commonly used or weak passwords and temporary lockout after multiple incorrect login attempts to mitigate the risk of brute-force or guessing attacks.

Remediation

SOLVED: The **InFlux Technologies team** resolved and updated the password complexity policy to include a recommendation for stronger passwords. While strict enforcement is not applied, users are now informed of the risks when selecting weaker passwords. Additionally, progressive delay mechanisms have been considered to prevent excessive brute-force attempts while maintaining user convenience.

Remediation Hash

https://github.com/RunOnFlux/ssp-wallet/pull/376

8.25 HAL-11 - IOS - BACKGROUND SCREEN CACHING

// LOW

Description

Background screen caching occurs when an iOS application’s screen is captured and displayed as a preview in the App Switcher view. While this feature provides a smooth user experience, it poses a potential security risk when sensitive information, such as user details, transaction details, or private data, is displayed. If an attacker gains access to the cached preview (e.g., via a compromised or shared device), they may extract this sensitive data, violating user privacy and security expectations.

Proof of Concept

SSP Key

Select Chain

Bitcoin Extended Public Key:

Extended Public key is used to derive all your Bitcoin public keys from which addresses are constructed. This is a public part, safe to share, does not grant access to funds.

*** *** *** *** *** ***

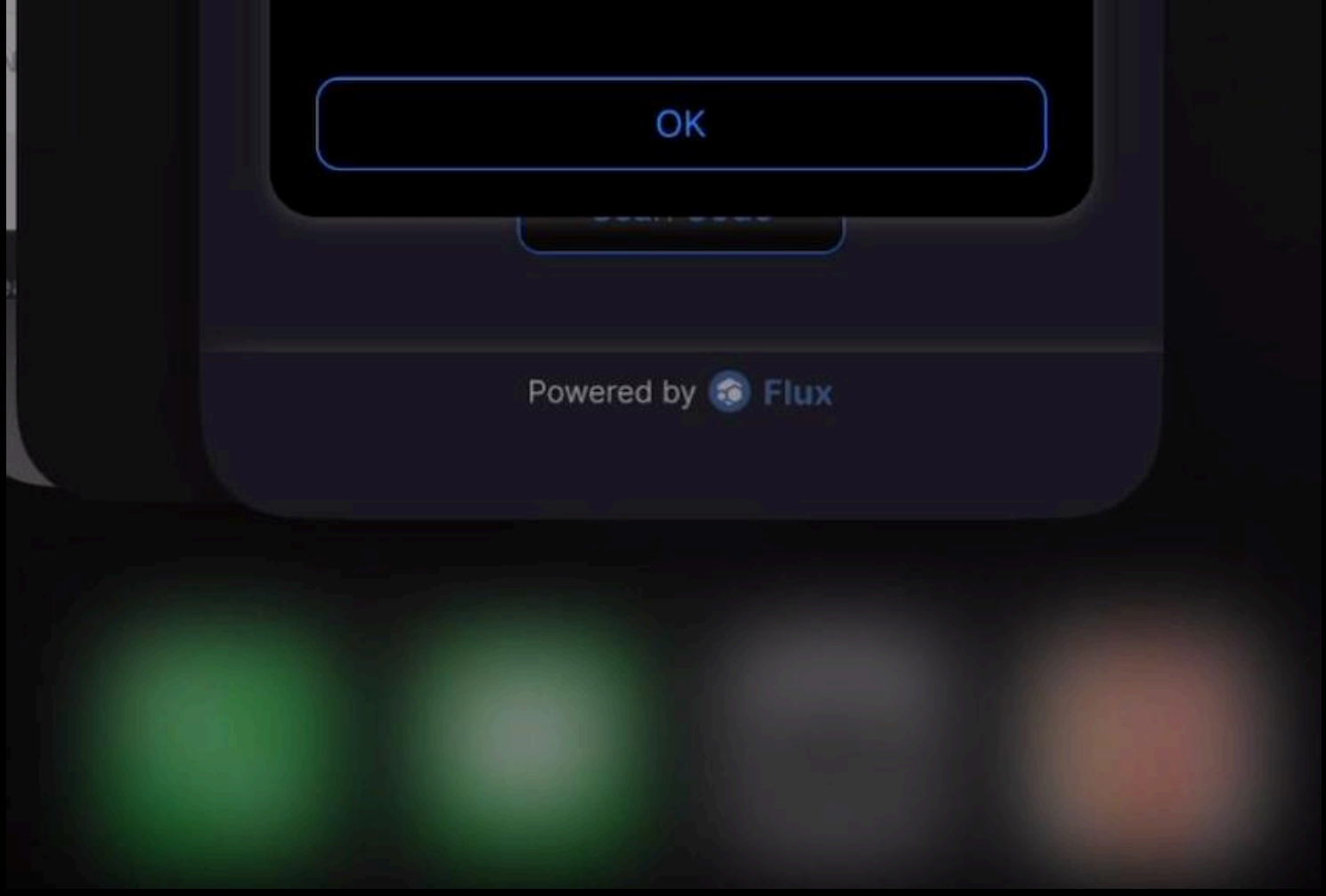
Bitcoin Extended Private Key:

Extended Private key is used to derive all your Bitcoin private keys that control your public keys - addresses. This is a private part, do NOT ever share as it grants access to funds.

*** *** *** *** *** ***

SSP Key Mnemonic Seed Phrase:

Seed Phrase is the most important part of your wallet. It is used to derive all your extended private, public keys. Please write it down and keep it safe. If you lose your Seed Phrase, you will lose access to your funds. Do NOT ever share with anyone as it grants full control over SSP Key part.



Score

Impact: 2

Likelihood: 2

Recommendation

To protect sensitive information from being visible in the App Switcher, implement measures to obscure or block sensitive screens when the app transitions to the background. Use a placeholder or blurred background to mask private data, ensuring it is not exposed to unauthorized access. This enhances privacy and security, particularly on shared or unattended devices.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by adding a blurred overlay.

Remediation Hash

d2d5a0eacd3332d1743cf9aaf5f36068687630a9

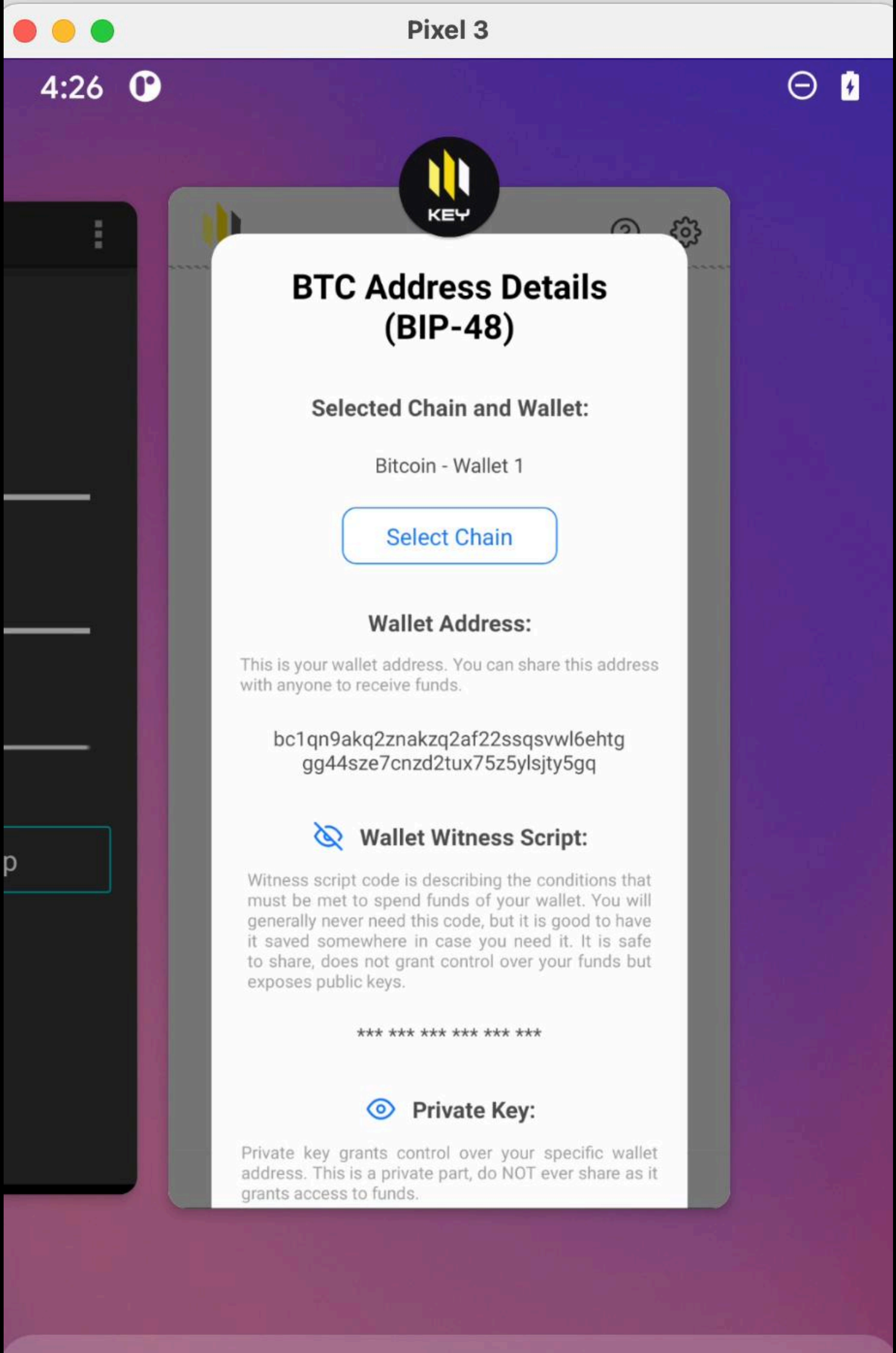
8.26 HAL-19 - ANDROID - BACKGROUND SCREEN CACHING

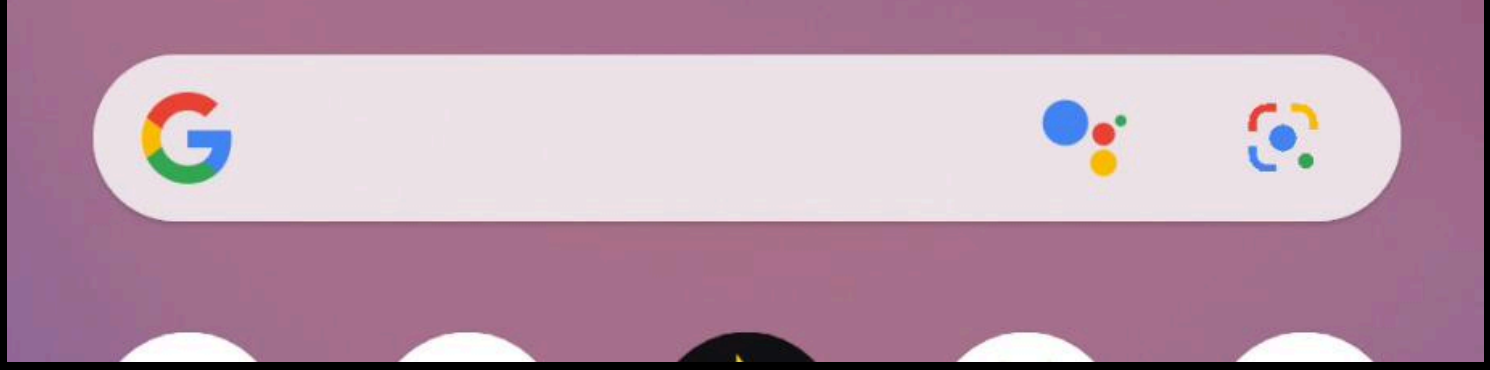
// LOW

Description

Background screen caching refers to the process where an application’s screen is captured and displayed as a preview in the recent apps/task manager view. While this feature provides a seamless user experience, it poses a potential security risk when sensitive information, such as user details, transaction details, or other private data, is displayed. If an attacker gains access to the cached preview (e.g., via a compromised or shared device), they can extract this sensitive data.

Proof of Concept


Pixel 3
4:26
KEY
BTC Address Details (BIP-48)
Selected Chain and Wallet:
Bitcoin - Wallet 1
Select Chain
Wallet Address:
This is your wallet address. You can share this address with anyone to receive funds.
bc1qn9akq2znakzq2af22ssqsvwl6ehtg gg44sze7cnzd2tux75z5ylsjty5gq
Wallet Witness Script:
Witness script code is describing the conditions that must be met to spend funds of your wallet. You will generally never need this code, but it is good to have it saved somewhere in case you need it. It is safe to share, does not grant control over your funds but exposes public keys.
*** *** *** *** *** ***
Private Key:
Private key grants control over your specific wallet address. This is a private part, do NOT ever share as it grants access to funds.



Score

Impact: 2

Likelihood: 2

Recommendation

To reduce the risk of sensitive information being exposed when the app transitions to the background, implement mechanisms to obscure the app screen. Use the FLAG_SECURE window flag for sensitive activities or fragments to prevent their content from being captured or displayed in the Android Overview (Recent Apps) screen. Additionally, override the onPause() or onStop() methods to display a placeholder view, blur sensitive content, or show a blank screen when the app moves to the background.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by implementing checks against the identified risk.

Remediation Hash

https://github.com/RunOnFlux/ssp-key/pull/76

8.27 HAL-24 - ANDROID - TRANSACTION DATA EXPOSED IN LOGS

// LOW

Description

The application logs sensitive information, including wallet metadata and push notification tokens, in plaintext within the device logs. Logging such information introduces a potential security risk, particularly in shared or compromised environments. If a device is rooted, or if an attacker gains physical or logical access to the device, these logs could be accessed to retrieve sensitive data. Such exposure may allow attackers to impersonate push notifications, monitor user activity, or compromise the wallet, leading to unauthorized access and potential financial loss.
Although modern Android versions restrict third-party applications from accessing logs directly, this does not eliminate the risk entirely. Rooted devices, development builds with debugging enabled, or shared test environments may still expose logs to unintended parties. Therefore, logging sensitive information unnecessarily increases the attack surface and weakens the overall security posture of the application.

Proof of Concept

```
01-03 20:51:36.782 21427 21843 D Camera2CameraImpl: {Camera@9d66790[id=0]} Transitioning camera internal state: CLOSING --> INITIALIZED
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: Recalculating open cameras:
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: Camera                                         State
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: ----------------------------------------------------------------------
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: Camera@9bc79a[id=1]                            UNKNOWN
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: Camera@d19754[id=2]                            UNKNOWN
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: Camera@9d66790[id=0]                           CLOSED
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: Camera@9ff45f9[id=3]                           UNKNOWN
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: ----------------------------------------------------------------------
01-03 20:51:36.782 21427 21843 D CameraStateRegistry: Open count: 0 (Max allowed: 1)
01-03 20:51:36.782 21427 21843 D CameraStateMachine: New public camera state CameraState{type=CLOSED, error=null} from CLOSED and null
01-03 20:51:36.782 21427 21843 D CameraStateMachine: Publishing new public camera state CameraState{type=CLOSED, error=null}
01-03 20:51:40.257 21427 21483 I ReactNativeJS: Open Authentication
01-03 20:51:40.280 21427 21483 I ReactNativeJS: entered auth
01-03 20:51:40.287 21427 21483 I ReactNativeJS: Biometrics is supported
01-03 20:51:40.290 21427 21484 W unknown:ReactNative: Attempt to set local data for view with unknown tag: -1
01-03 20:51:40.290 21427 21484 I chatty  : uid=10254(io.runonflux.sspkey) mqt_native_modu identical 1 line
01-03 20:51:40.290 21427 21484 W unknown:ReactNative: Attempt to set local data for view with unknown tag: -1
01-03 20:51:40.550 21427 21483 I ReactNativeJS: Initiate Fingerprint
01-03 20:51:43.217 21427 21483 I ReactNativeJS: successful biometrics provided
01-03 20:51:43.217 21427 21483 I ReactNativeJS: true
01-03 20:51:43.217 21427 21483 I ReactNativeJS: authentication modal close.
01-03 20:51:43.227 21427 21483 I ReactNativeJS: Approve
01-03 20:51:43.383 21427 21483 I ReactNativeJS: { chain: 'fluxTestnet',
01-03 20:51:43.383 21427 21483 I ReactNativeJS:   walletIdentity: '154qEe3F9YhxKoJc43v7DQjWFJEq1cVkYo',
01-03 20:51:43.383 21427 21483 I ReactNativeJS:   keyXpub: 'tpubDEXy787qqe3PShzd9pzYxoo2p4CyBuToFmr1aTJUKNbvt6Zi1tp7Z1WMZSgrpJ19mHkRifzfXVbR1uCXtMaqzP5d62gpaeVRo7KGtDvPoPU',
01-03 20:51:43.383 21427 21483 I ReactNativeJS:   wkIdentity: 'bc1qyyjmzmg8pfn6e8d0eehqujy6rf4yvmh794kg27q5txsway7mkyjs8xvy08',
01-03 20:51:43.383 21427 21483 I ReactNativeJS:   keyToken: 'fSVgFE9wTvex0iI9NF_B97:APA91bHepQanIgoCcvEPeKI3JzS9djDCf53iQth2YdzLQrEWaBntwTgGXtZYUaoHUiO5vd0l3oYoGiG57bLz-EDO_spSmbZVG2SjtkDZwdqfmBg911na_2o' }
```

Score

Impact: 2

Likelihood: 2

Recommendation

To reduce the risk of sensitive data exposure, avoid logging confidential information such as private keys, wallet identifiers, and authentication tokens. Ensure logs are sanitized to remove sensitive fields and disable verbose logging in production environments.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by implementing appropriate checks against the identified risk.

Remediation Hash

https://github.com/RunOnFlux/ssp-key/pull/76

8.28 HAL-21 - IOS - APPLICATION ALLOWS SCREENSHOTS

// LOW

Description

The iOS application currently allows users to take screenshots of sensitive screens, such as those displaying private keys, seed phrases, or other confidential data. This behavior poses a security risk as screenshots can be inadvertently shared, stored in cloud backups, or accessed by unauthorized parties. On shared or compromised devices, an attacker could gain access to sensitive information by reviewing stored screenshots.
A user can take a screenshot of sensitive data displayed in the application, such as the seed phrase or private key, without any restriction. These screenshots are stored in the device's photo gallery and potentially synced with cloud services, making the data more vulnerable to unauthorized access.

Score

Impact: 2

Likelihood: 2

Recommendation

To prevent sensitive data from being captured via screenshots or screen recordings, implement screen security features in the application. Use UIApplication.shared.isScreenCaptured to detect when the screen is being captured and take appropriate action, such as displaying a warning or obscuring sensitive content. Additionally, implement privacy protections such as blurring sensitive views using UIWindowLevelSecure when the app goes into the background or during app switching. These measures will help reduce the risk of sensitive information being exposed through screenshots or recordings.

Remediation

FUTURE RELEASE: The **InFlux Technologies team** will address the issue in the future builds of the applications.

8.29 HAL-20 - ANDROID - APPLICATION ALLOWS SCREENSHOTS

// LOW

Description

The application currently allows users to capture screenshots of sensitive screens displaying confidential information such as mnemonic seed phrase, private keys. This poses a security risk, as such screenshots can expose private financial information that could be misused if shared or accessed by unauthorized parties. This risk is particularly critical in scenarios where devices are shared, stolen, or compromised, as screenshots stored on the device can be accessed by rogue applications or attackers. Allowing screenshots increases the potential for sensitive financial data to be inadvertently leaked, resulting in a compromise of user privacy and security.

Proof of Concept

1. Open the application and navigate to a screen displaying sensitive information, such as mnemonic phrase or private key.
2. Capture a screenshot using the device's screenshot feature.
3. Observe that the screenshot allowed without any restrictions.

Score

Impact: 2

Likelihood: 2

Recommendation

To prevent sensitive information from being captured through screenshots, enable the `FLAG_SECURE` flag on windows displaying sensitive content. This can be done by calling `getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE, WindowManager.LayoutParams.FLAG_SECURE)` in the `onCreate()` method of relevant activities. This ensures that screenshots, screen recordings, and streaming are disabled for the protected screens, safeguarding user data.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by implementing checks against the identified risk.

Remediation Hash

https://github.com/RunOnFlux/ssp-key/pull/76

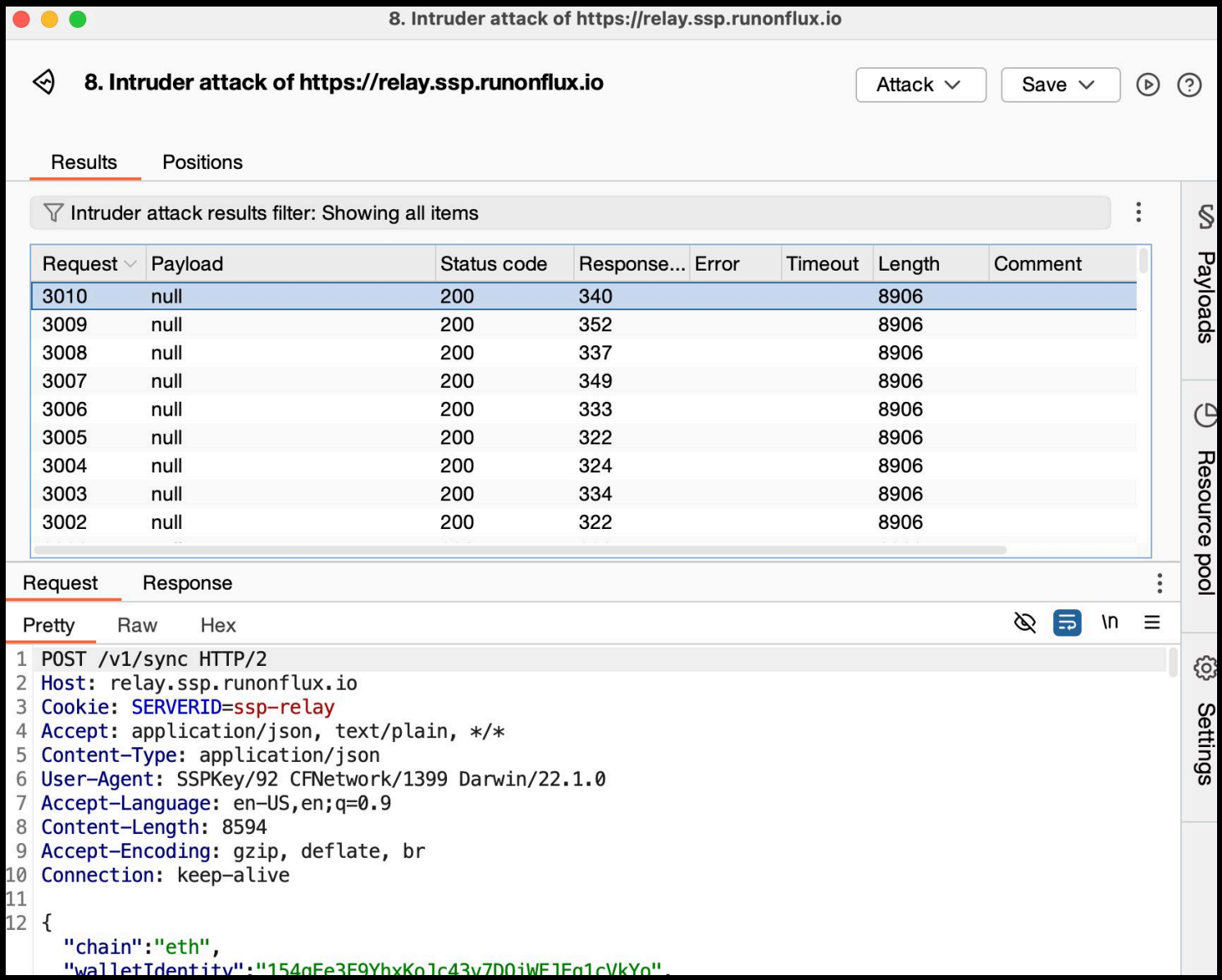
8.30 HAL-23 - API - LACK OF RATE LIMITATION ON SSP RELAY ENDPOINTS

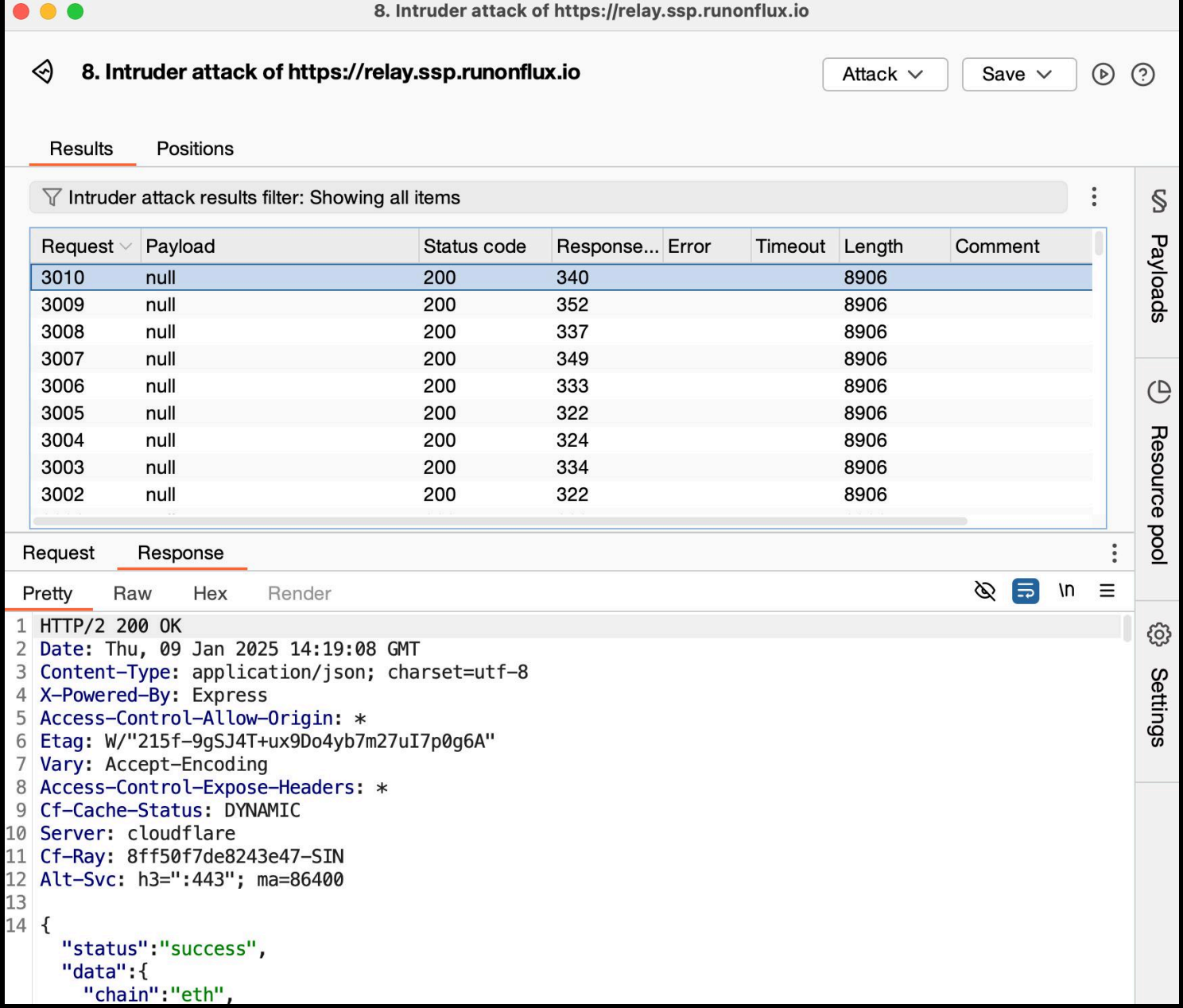
// MEDIUM

Description

The SSP Relay server facilitates communication between the SSP Wallet and SSP Key by relaying requests for synchronization, signing, and other wallet operations. The SSP Wallet can function offline, and critical wallet operations, including signing transactions and synchronizing keys, can be completed without relying on the SSP Relay server. This reduces the reliance on the relay server, minimizing the potential impact of abuse. However, despite the reduced risk, the absence of rate-limiting leaves the endpoints theoretically susceptible to denial-of-service (DoS) or brute-force attacks, potentially impacting availability.

Proof of Concept





Score

Impact: 3

Likelihood: 3

Recommendation

Implementing rate limiting on critical SSP Relay endpoints is still recommended to prevent potential abuse or denial-of-service attacks. Applying a reasonable request rate limit (e.g., X requests per minute per IP or per wallet identity) can help mitigate the risk of endpoint overloading.

Remediation

SOLVED: The **InFlux Technologies team** resolved the issue by implementing the rate limitation protection.

Remediation Hash

`a17bd2596fb89c3a851b6ff8dab334055895031f`

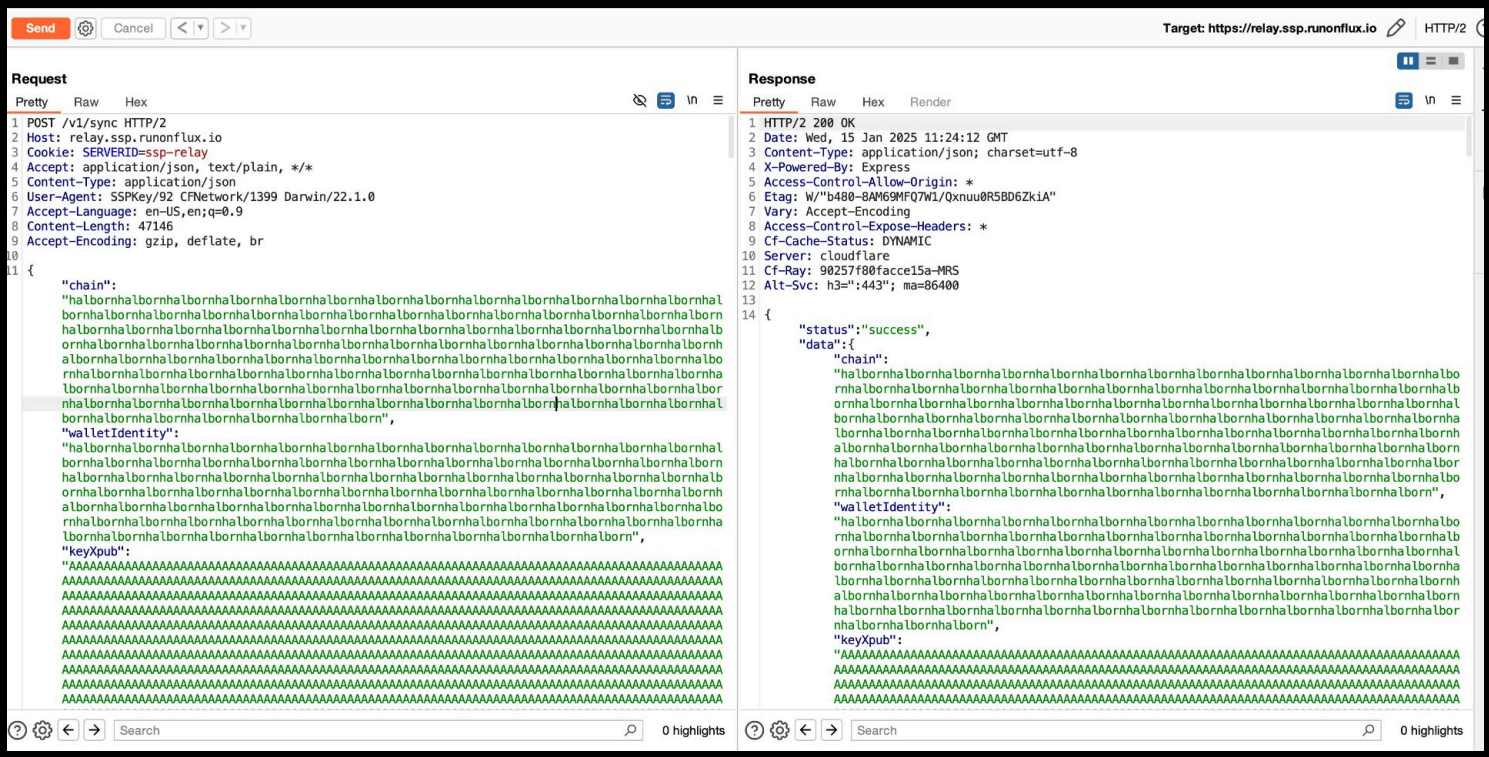
8.31 HAL-27 - API - LACK OF DATA SANITIZATION AND VALIDATION OF LIMITS

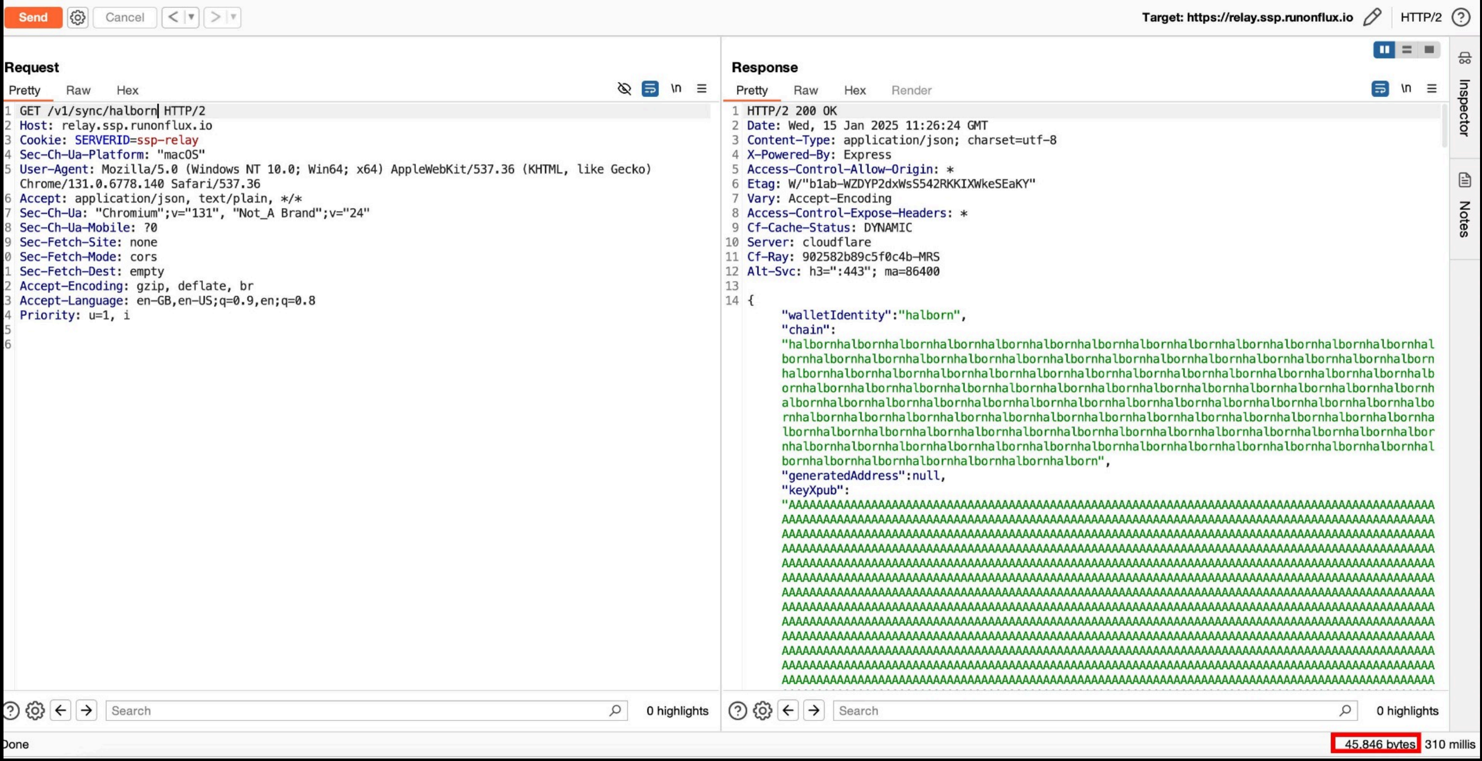
// MEDIUM

Description

The SSP relay API endpoint have insufficient data sanitization and lack of proper input size validation. Specifically, the parameters can accept excessively large input without any restrictions. This absence of data validation poses a potential risk, as it may lead to potential denial-of-service (DoS) attacks or resource exhaustion by sending large payloads, impacting the availability and performance of the service. Attackers can exploit this weakness by crafting large payloads to consume excessive server resources, degrade the user experience, or even crash the service. Additionally, the lack of input sanitization might expose the application to further risks, such as injection-based attacks.

Proof of Concept





Score

Impact: 3

Likelihood: 3

Recommendation

It is recommended to implement robust input validation and sanitization for all incoming data at the server level. Specifically:

- Enforce strict size limits on all fields to prevent excessively large payloads from being processed.
- Validate the data type and format of all parameters to ensure they meet expected criteria.
- Sanitize input data to mitigate potential injection attacks and ensure safe handling of user-provided data.
- Return appropriate error messages for inputs that exceed the defined limits, and log such attempts for further investigation.
- Consider implementing rate limiting to prevent abuse through high-frequency requests with large payloads.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by implementing limit and checks on the parameters.

Remediation Hash

a17bd2596fb89c3a851b6ff8dab334055895031f

8.32 HAL-05 - IOS - INSECURE ACCESSIBILITY ATTRIBUTES IN KEYCHAIN STORAGE

// MEDIUM

Description

The application uses insecure Keychain accessibility attributes for storing sensitive data, such as cryptographic keys, Firebase tokens, and PINs. Attributes like `AlwaysThisDeviceOnly`, `AfterFirstUnlockThisDeviceOnly`, and `WhenUnlocked` do not provide the necessary security for critical data. This allows access to sensitive information under less secure conditions, such as when the device is locked or after a single unlock, increasing the risk of data compromise.

Proof of Concept

```
io.runonflux.sspkey on (iPhone: 16.1.1) [usb] # ios keychain dump
Note: You may be asked to authenticate using the devices passcode or TouchID
Save the output by adding `--json keychain.json` to this command
Dumping the iOS keychain...
Created                    Accessible                      ACL   Type      Account                                                   Service                                     Data
-------------------------  ------------------------------  ----  --------  --------------------------------------------------------  ------------------------------------------  ----------
--------------------------------------------------------------------------------------------------------------------------------------
2025-01-01 14:48:39 +0000  AfterFirstUnlockThisDeviceOnly  None  Password  io.runonflux.sspkey                                       233280865272:*
2025-01-01 14:48:37 +0000  AfterFirstUnlockThisDeviceOnly  None  Password  io.runonflux.sspkey                                       com.google.iid.checkin                      5233466188
429178126|952631838458496453
2024-12-31 20:57:50 +0000  AfterFirstUnlockThisDeviceOnly  None  Password  1:233280865272:ios:8218b3f3e444651b2d0c0d__FIRAPP_DEFAULT  com.firebase.FIRInstallations.installations
2025-01-01 14:48:36 +0000  AlwaysThisDeviceOnly            None  Password  _pfo                                                      io.runonflux.sspkey                         1
2024-12-31 20:55:59 +0000  WhenUnlocked                    None  Password  ssp_key_pw                                                                                            1111
```

Score

Impact: 3

Likelihood: 3

Recommendation

It is recommended to review the keychain accessibility attributes, sensitive data such as cryptographic keys, tokens, and PINs should be stored using the `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` attribute. This ensures that data is accessible only when the device is unlocked and protected by a secure passcode. Avoid using weak attributes like `AlwaysThisDeviceOnly` or `WhenUnlocked` for critical information where necessary.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by opting for `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`, ensuring accessibility when the device is unlocked while balancing user experience with security.

Remediation Hash

dc10902f97e480a36334ecfe63b3d6e4bc13fdcc

8.33 HAL-22 - BE - POTENTIAL RISK OF SENSITIVE DATA EXPOSURE THROUGH CLIPBOARD

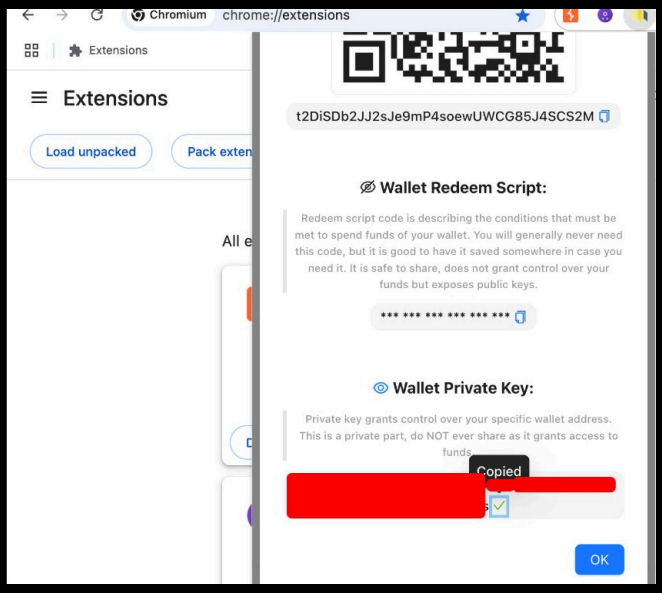
// MEDIUM

Description

The wallet application facilitates copying sensitive data, specifically mnemonic and private key passphrases, to the clipboard. This functionality presents a significant security risk, as clipboard data can be accessed both locally and remotely by unauthorized processes or malicious web pages. Attackers can exploit this vulnerability by leveraging scripts or pages designed to capture clipboard contents, thereby compromising the confidentiality of critical information.
Furthermore, in multi-device environments where clipboard sharing is enabled (such as between smartphones, tablets, and laptops), sensitive data copied to the clipboard can be inadvertently exposed across multiple devices. This significantly increases the attack surface, as an attacker gaining access to any linked device can retrieve the copied mnemonic or private key. The resulting risk of unauthorized access or asset theft underscores the need for stringent handling of sensitive data within the application.

Proof of Concept

```
max in ~/Desktop/M-PoCs λ python3 clip-monitor.py
Leaked private key:
[redacted]
```



Score

Impact: 3

Likelihood: 3

Recommendation

It is recommended to avoid providing a direct copy-to-clipboard feature for sensitive information such as mnemonic and private key passphrases. Instead, consider implementing a secure display mechanism that requires users to manually input or write down the passphrase. If clipboard functionality is essential for user experience, implement automatic clipboard clearing after a short period to reduce exposure time. Additionally, display a warning when sensitive information is copied, advising users to clear their

clipboard and avoid copying data in shared or multi-device environments where clipboard data can sync across devices. These measures will significantly reduce the risk of unauthorized access to sensitive information.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by splitting the seed phrase and private key into multiple parts during copying, which prevents direct exposure. Additional warnings and explanations have also been added to inform users about the associated risks of copying these sensitive data.

8.34 HAL-03 - BE - UNRESTRICTIVE CONTENT-SECURITY-POLICY (CSP)

// MEDIUM

Description

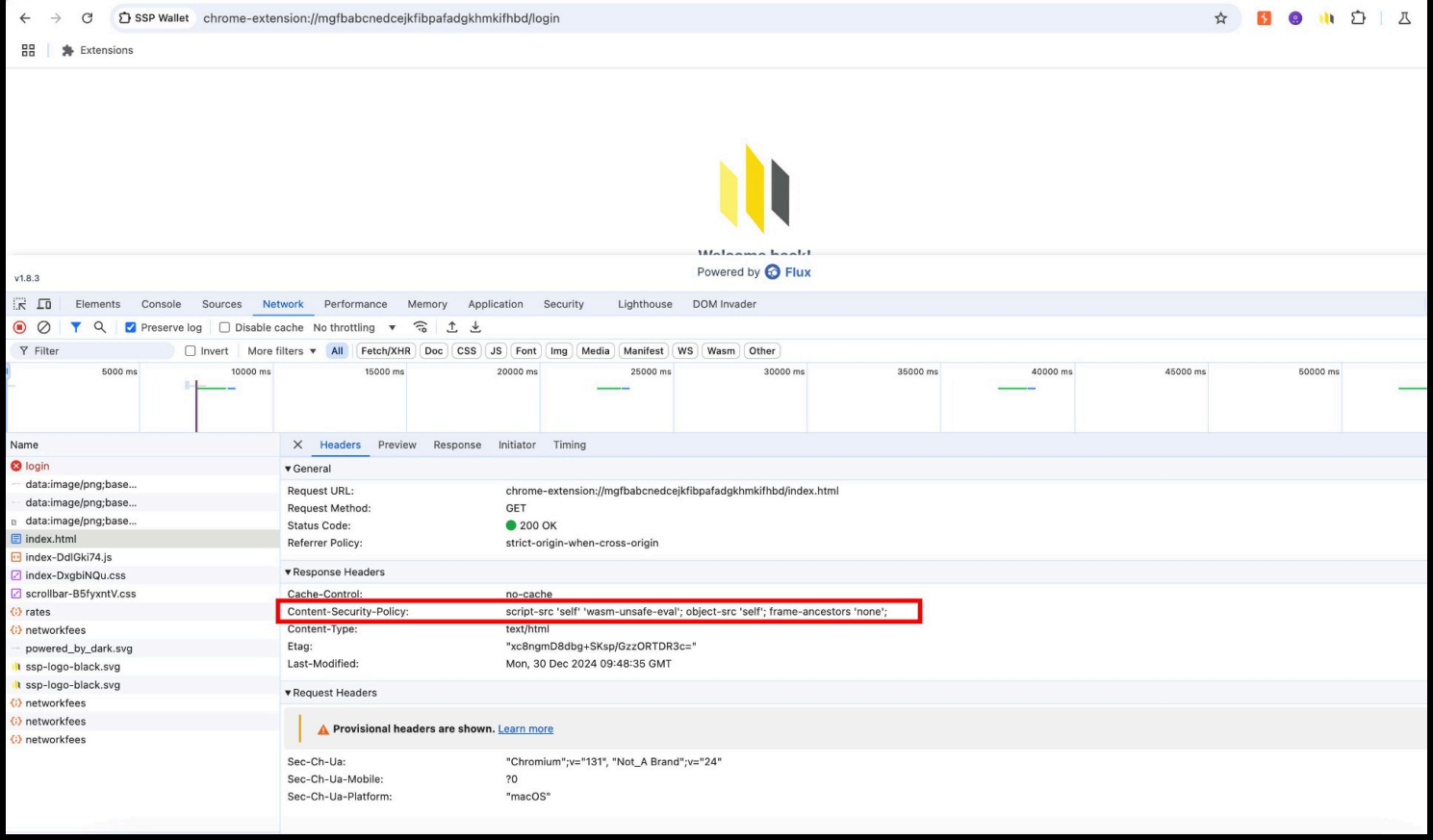
The wallet browser extension has a partially restrictive declared Content-Security-Policy (CSP) that could be improved to enhance security. Specifically, it fails to define a `default-src` directive, which acts as a fallback policy ensuring that no malicious resources are loaded from untrusted sources that have not been explicitly declared. The absence of this fallback increases the risk of loading unintended content from unauthorized origins.

Moreover, the `object-src` directive is currently set to `'self'`, which permits the use of legacy HTML elements such as `<object>, <embed>,` and `<applet>`. These elements are prone to security risks and are no longer receiving new standardized security features (such as the sandbox or allow attributes for `<iframe>`). Therefore, it is recommended to explicitly set `object-src 'none'` to eliminate this potential attack vector unless explicitly required.

Lastly, several important directives that do not inherit from default-src are not defined, leaving additional areas unprotected. These missing directives include:

- base-uri
- form-action
- plugin-types
- report-uri
- sandbox
- reflected-xss
- referrer

Proof of Concept




```
  "content_security_policy": {
    "extension_pages": "script-src 'self' 'wasm-unsafe-eval'; object-src 'self';
frame-ancestors 'none';"
  },
```

Score

Impact: 3

Likelihood: 3

Recommendation

It is recommended to add the `default-src 'none'` directive to your CSP as a fallback. It is also recommended to set the `object-src` directive to none if possible since the elements it controls are considered legacy, and to set the missing directives which do not inherit from the `default-src` directive.

Remediation

SOLVED: The **InFlux Technologies team** addressed the issue by implementing checks against the identified risk.

References

https://mikewest.org/2011/10/secure-chrome-extensions-content-security-policy/
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/default-src

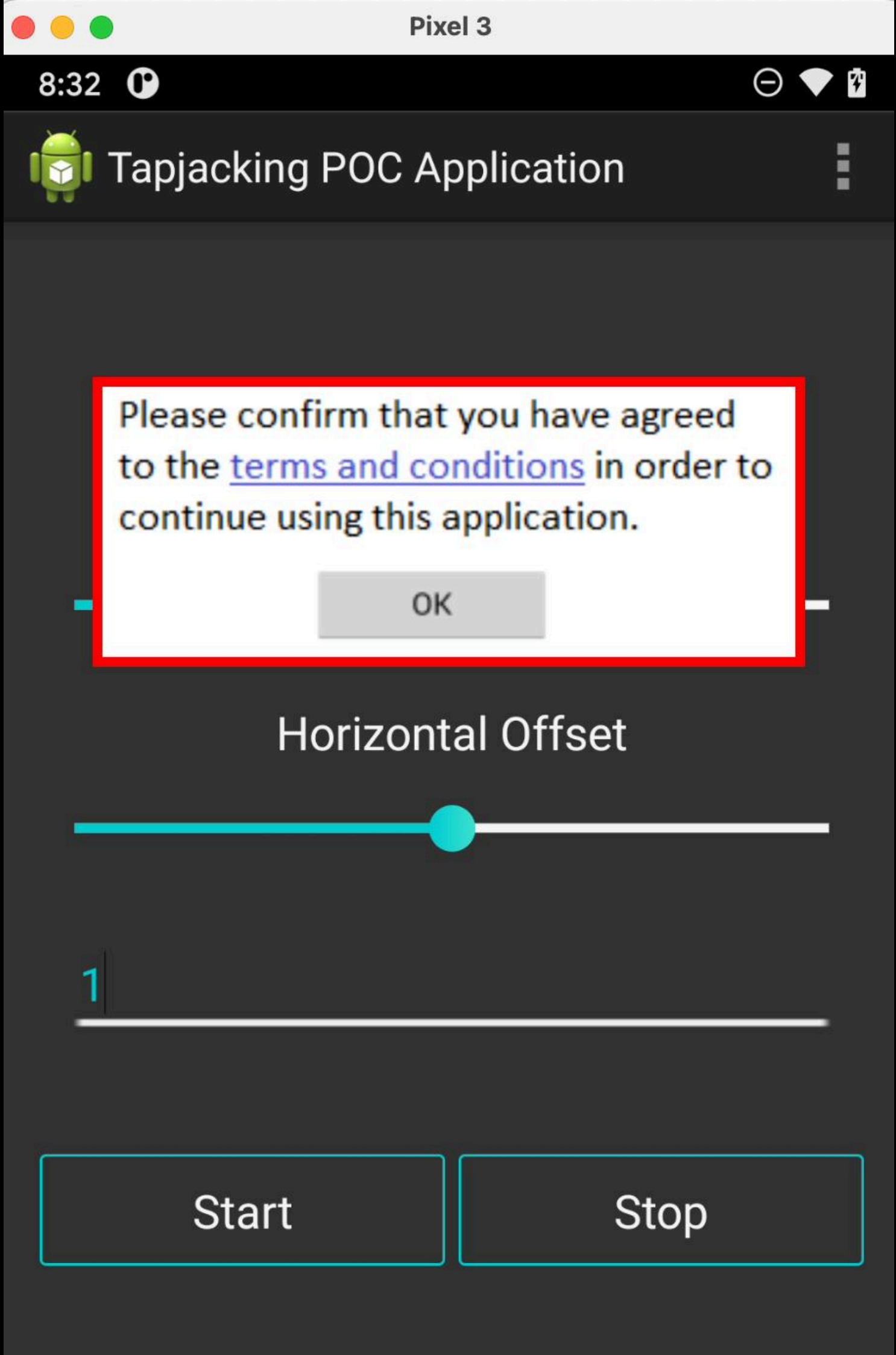
8.35 HAL-32 - ANDROID - RISK OF OVERLAY ATTACK

// MEDIUM

Description

Overlay attacks exploit the ability to display a user interface element on top of other applications, which malicious apps can use to trick users into performing unintended actions. This is often achieved by displaying a fake UI or dialog over a legitimate app, leading to risks such as credential theft, like user password and PIN, or unauthorized actions.
For instance, a malicious app could display a login form over a legitimate app, tricking users into entering their credentials, mnemonics, and Private keys. Android's `SYSTEM_ALERT_WINDOW` permission enables apps to display overlays, and without proper protections, your application could become vulnerable to these attacks.

Proof of Concept


Pixel 3
8:32
Tapjacking POC Application
Please confirm that you have agreed to the terms and conditions in order to continue using this application.
OK
Horizontal Offset
1
Start
Stop

Pixel 3

8:34

Please confirm that you have agreed to the terms and conditions in order to continue using this application.

OK

No pending actions.

Refresh

Score

Impact: 3

Likelihood: 3

Recommendation

To prevent overlay attacks, implement the following best practices:
1. **Detect and Block Overlays**: Use the TYPE_APPLICATION_OVERLAY or TYPE_SYSTEM_ALERT APIs to detect overlays when sensitive screens are displayed. In critical workflows (e.g., entering credentials), validate that no overlays are active by checking for apps with SYSTEM_ALERT_WINDOW permissions.
2. **Secure Sensitive UI**: Protect sensitive screens by detecting active overlays and preventing interaction if any are present. Use the getWindow().setFlags() method to set secure flags:

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
                     WindowManager.LayoutParams.FLAG_SECURE);
```

3. **User Awareness**: Warn users if overlays are detected while performing sensitive actions, such as entering credentials or completing a transaction.
4. **Limit Clickable Areas**: Employ clickjacking prevention techniques by disabling `onClick` events when overlays are detected.
5. **Restrict Permissions**: Encourage users to limit the SYSTEM_ALERT_WINDOW permission for untrusted apps via device settings.

Remediation

PARTIALLY SOLVED: The **InFlux Technologies team** partially addressed the issue by implementing security flags, and further communicated that edge cases related to this will be further addressed in future builds of the application.

Remediation Hash

https://github.com/RunOnFlux/ssp-key/pull/79

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.