Uniswap Mobile Wallet

Fix Review

December 12, 2022

Prepared for:
Padmini Pyapali
Uniswap

Prepared by: **Alexander Remie**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

Table of Contents

Executive Summary

Engagement Overview

Uniswap engaged Trail of Bits to review the security of its mobile wallet. From August 8 to August 19, 2022, a team of four consultants conducted a security review of the client-provided source code, with four person-weeks of effort. Details of the project's scope, timeline, test targets, and coverage are provided in the original audit report.

Uniswap contracted Trail of Bits to review the fixes implemented for issues identified in the original report. From October 24 to October 26, 2022, a team of one consultant conducted a review of the client-provided source code, with three person-days of effort.

Summary of Findings

The original audit uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the original findings is provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	7
Medium	1
Low	1
Informational	3
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Configuration	1
Cryptography	4
Data Exposure	4
Data Validation	1
Error Reporting	1
Patching	1

Overview of Fix Review Results

Uniswap has sufficiently addressed most of the issues described in the original audit report.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineers were associated with this project:

Alexander Remie, Consultant
alexander.remie@trailofbits.com

Bo Henderson, Consultant
bo.henderson@trailofbits.com

Emilio López, Consultant
emilio.lopez@trailofbits.com

Tjaden Hess, Consultant
tjaden.hess@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
August 8, 2022	Pre-project kickoff call
August 15, 2022	Status update meeting #1
August 22, 2022	Delivery of report draft; report readout meeting
September 8, 2022	Delivery of final report
December 12, 2022	Delivery of fix review

Project Methodology

Our work in the fix review included the following:

- A review of the findings in the original audit report
- A manual review of the client-provided source code and configuration material

Project Targets

The engagement involved a review of the fixes implemented in the following target.

Uniswap Mobile Wallet

Repository	https://github.com/Uniswap/mobile
Version	`c8445da435f19b671bd4eadd763f63e8b97b284d`
Type	React Native, Typescript, Swift
Platform	Mobile (iOS)

Summary of Fix Review Results

The table below summarizes each of the original findings and indicates whether the issue has been sufficiently resolved.

ID	Title	Status
1	iOS client is susceptible to URI scheme hijacking	Resolved
2	The iOS client does not disable custom keyboards	Resolved
3	Encrypted iCloud backups use low-entropy keys	Resolved
4	Users are allowed to create unencrypted iCloud backups	Resolved
5	Remote Timing Side Channel in WalletConnect Library	Unresolved
6	Use of libraries with known vulnerabilities	Partially Resolved
7	Sending funds to user-owned addresses can cause spurious warnings	Resolved
8	WalletConnect v1 reuses cryptographic keys for multiple primitives	Unresolved
9	Credentials checked into source control	Partially Resolved
10	NFTs with SVG images are rendered as HTML	Unresolved
11	Application does not exclude keychain items from iCloud and iTunes backups	Resolved
12	ShakeBugs may leak mnemonic	Partially Resolved

13	Use of improperly pinned GitHub Actions in Testflight build	**Resolved**

Detailed Fix Review Results

1. iOS client is susceptible to URI scheme hijacking	
Status: **Resolved**	
Severity: **High**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-UNIMOB-001
Target: `ios/Uniswap/Info.plist`	

Description

The Uniswap mobile app defines the `uniswap://` URI scheme for receiving messages from other apps on the device. URI schemes can be hijacked by another app if the malicious app registers the same scheme and is also installed on the device. Consequently, a rogue app could receive messages sent via URI schemes intended for the Uniswap mobile app.

```
32    <key>CFBundleURLName</key>
33    <string>uniswap</string>
34    <key>CFBundleURLSchemes</key>
35    <array>
36          <string>uniswap</string>
37    </array>
```

Figure 1.1: `ios/Uniswap/Info.plist`

Fix Analysis

This issue has been resolved. The `uniswap://` deep link is no longer supported in the application and is replaced by the use of Universal Links. This prevents the aforementioned issue.

2. The iOS client does not disable custom keyboards

Status: **Resolved**	
Severity: **Low**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-UNIMOB-002
Target: Uniswap iOS application	

Description

The Uniswap mobile app client does not disable custom keyboards. Since iOS 8, users have been able to replace the system's default keyboard with custom keyboards that can be used in any application. Custom keyboards can—and very frequently do—log and exfiltrate the data that users enter.

Custom keyboards are not enabled when users type into "secure" fields (such as password fields). However, they could log all of a user's keystrokes in regular fields, such as those in which users type their personal information.

Fix Analysis

This issue has been resolved by disabling the use of custom keyboards in the application.

3. Encrypted iCloud backups use low-entropy keys

Status: **Resolved**	
Severity: **High**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-UNIMOB-003
Target: `src/features/CloudBackup`	

Description

During onboarding and new wallet creation, the iOS wallet application presents the user with two backup options: a manual backup of the wallet mnemonic and an iCloud backup. The iCloud backup may be optionally encrypted using a six-digit numeric pin. If the option to encrypt is selected, the wallet derives an AES-GCM encryption key from the pin via 310000 rounds of PBKDF2 with a random salt.

```
export const PIN_LENGTH = 6
```

Figure 3.1: src/features/CloudBackup/cloudBackupSlice.ts#L11

```
func encrypt(secret: String, password: String, salt: String) throws -> String {
  let key = try keyFromPassword(password: password, salt: salt)
  let secretData = secret.data(using: .utf8)!

  // Encrypt data into SealedBox, return as string
  let sealedBox = try AES.GCM.seal(secretData, using: key)
  let encryptedData = sealedBox.combined
  let encryptedSecret = encryptedData!.base64EncodedString()

  return encryptedSecret
}
```

Figure 3.2: EncryptionHelper.swift#L18-L28

If an attacker were to obtain access to the user's iCloud account and retrieve the encrypted wallet file, they would be able to easily decrypt the file due to the low entropy of the six-digit pin. While password-based key derivation functions provide a linear slowdown to attackers brute-forcing passwords, there are only one million six-digit pins. At an extremely conservative estimate of one CPU-second per password attempt, an attacker could brute-force the password in one million CPU-seconds; at the time of writing, this would cost about $650 on AWS's serverless compute service.

Because PBKDF2 is not GPU-resistant, an attacker could potentially try all of the possible passwords in minutes to hours on a single consumer GPU.

Fix Analysis

This issue has been resolved. The use of a PIN to encrypt the wallet backup has been replaced by the use of a password (with minimum length 8). The KDF algorithm used is argon2id, and the parameters are based on recommended values from the Argon2 RFC and Appendix E in the final report of this audit. Additionally, a time-based lockout mechanism has been implemented for when incorrect passwords are entered.

4. Users are allowed to create unencrypted iCloud backups

Status: **Resolved**	
Severity: **High**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-UNIMOB-004
Target: Uniswap iOS wallet iCloud backups	

Description

During user onboarding and wallet creation, users are given the option to back up a copy of their mnemonic private key to iCloud. Users are prompted to enter a six-digit pin for encryption (see TOB-UNIMOB-003) with the option to skip encryption. Skipping encryption prompts a warning dialogue that notifies the user that their keys will be unprotected if their iCloud account is compromised.

The option to skip encryption, even with a warning, presents a footgun to users. iCloud accounts have a large attack surface and can be compromised (e.g., through phishing campaigns or zero-day server exploits).

Users may use the manual seed phrase backup to store their wallet as securely or insecurely as they choose, but the default cloud backup flow should be secure in all circumstances.

Fix Analysis

This issue has been resolved. The option to have an unencrypted iCloud backup has been removed, and all iCloud backups are now encrypted.

5. Remote timing side channel in WalletConnect library

Status: **Unresolved**	
Severity: **Medium**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-UNIMOB-005
Target: `WalletConnectSwift/Sources/Internal/AES_256_CBC_HMAC_SHA256_Codec.swift`	

Description

WalletConnect is a protocol for relaying messages between DApps and user Wallets. Setup is accomplished by sharing a QR code specifying a symmetric private key as well as a Bridge node responsible for relaying requests and maintaining pub/sub queues. The Bridge node is designed as an untrusted intermediary that blindly passes encrypted and authenticated messages between the Wallet and DApp.

The WalletConnect v1 Specification requires that clients communicate using AES-CBC for encryption along with HMAC-SHA256 to authenticate the data. When implementing authentication via HMAC, it is important that the time taken to compare the computed HMAC tag with the tag attached to the message does not depend on the content of the computed tag. If the standard string comparison function is used, the comparison will exit on the first mismatching byte and thus via a timing channel reveal how many leading bytes of the message MAC match the correct MAC for that message.

For example, the Swift implementation of the WalletConnect library compares the computed MAC with the payload MAC using the default "==" function:

```
let hmac = try authenticationCode(key: keyData, data: payload.data.data +
payload.iv.data)
guard hmac == payload.hmac.data else {
    throw CodecError.authenticationFailed(cipherText)
}
```

Figure 5.1: Sources/Internal/AES_256_CBC_HMAC_SHA256_Codec.swift#L65

Although this is an issue in the upstream implementation of the `WalletConnectSwift` dependency and not in the Uniswap mobile wallet implementation, the vulnerability affects the mobile wallet by potentially allowing a malicious WalletConnect bridge to forge

message requests, including changing the token amounts in valid DApp requests and tampering with wallet responses to queries of balances and other blockchain states.

In order to exploit this side channel, the bridge needs to observe some timing-dependent behavior from the client. This can take many forms, including measuring the wallet response time to replayed messages.

This issue can be remediated in the WalletConnect library implementations via either of the following two methods:

1. Use a constant-time hash comparison routine from a cryptography library, such as https://developer.apple.com/documentation/cryptokit/hmac/3237468-isvalidauthenticationcode
2. Implement randomized double-HMAC blinding as described in https://paragonie.com/blog/2015/11/preventing-timing-attacks-on-string-comparison-with-double-hmac-strategy

For example, the Swift implementation in Figure 5.1 could be modified as in Figure 5.2:

```
let mask = [Int8](repeating: 0, count: 32)
let status = SecRandomCopyBytes(kSecRandomDefault, mask.count, &mask)
guard status == errSecSuccess else {
       throw CodecError.randomCopyFailed()
}
let hmac_inner = try authenticationCode(key: keyData, data: payload.data.data +
payload.iv.data)
let hmac_outer = try authenticationCode(key: keyData, data: hmac_inner)
let hmac_message = try authenticationCode(key: keyData, data: payload.hmac.data)
guard hmac_outer == hmac_message else {
    throw CodecError.authenticationFailed(cipherText)
}
```

Figure 5.2: Masked HMAC comparison example

Because this attack may be carried out against either party in the communication, the implementation must be patched in both the wallet and DApp.

Fix Analysis

This issue has not been resolved. However, because the issue is present in the upstream WalletConnect library, there is no way for Uniswap to address it; remediation must be handled by WalletConnect itself.

6. Use of libraries with known vulnerabilities

Status: **Partially Resolved**	
Severity: **Undetermined**	Difficulty: **Low**
Type: Patching	Finding ID: TOB-UNIMOB-006
Target: `package.json`	

Description

The codebase contains outdated dependencies affected by critical and high-risk vulnerabilities. We used `yarn audit` to detect a number of vulnerable packages that are referenced by the `yarn.lock` files. The most notable vulnerabilities are as follows:

- The `immer`, `minimist`, `simple-plist`, and `plist` dependencies contain critical vulnerabilities related to prototype pollution.
- The `shell-quote` package contains a critical vulnerability related to improper escapes that could allow arbitrary code execution (ACE) if the output from this package is passed to a shell.
- The `trim`, `terser`, `glob-parent`, `nth-check`, and `ansi-regex` dependencies are vulnerable to high-severity regular-expression denial-of-service (ReDoS) attacks.
- `node-fetch` and `follow-redirects` can expose sensitive data by leaking confidential HTTP headers while redirecting.
- `jpeg-js` is called in the `extractColors` utility function, potentially triggering an infinite loop leading to a denial of service (DoS).

Dependency	Vulnerability Type	Installed Version	Patched Version
`immer`	Prototype Pollution	8.0.1	9.0.6
`shell-quote`	ACE	1.6.1, 1.7.2	1.7.3
`hermes-engine`	Type Confusion	0.9.0	0.10.0
`minimist`	Prototype Pollution	1.2.5	1.2.6
`simple-plist`	Prototype Pollution	1.1.1	1.3.1
`plist`	Prototype Pollution	3.0.4	3.0.5
`trim`	ReDoS	0.0.1	0.0.3

`terser`	ReDoS	4.8.0	5.14.2
`glob-parent`	ReDoS	3.1.0	5.1.2
`node-fetch`	Data Exposure	1.7.3, 2.6.0, 2.6.1, 2.6.5	2.6.7
`nth-check`	ReDoS	1.0.2	2.0.1
`ansi-regex`	ReDoS	3.0.0, 4.1.0	3.0.1
`prismjs`	XSS	1.26.0	1.27.0
`follow-redirects`	Data Exposure	1.14.5	1.14.7
`async`	Prototype Pollution	2.6.3	2.6.4
`moment`	ReDoS, Path Traversal	2.29.1	2.29.4
`jpeg-js`	DoS	0.4.2	0.4.4

Figure 6.1: Dependencies with known critical- and high-severity vulnerabilities

Fix Analysis

This issue has been partially resolved. Some dependencies were updated, but no automated dependency auditing has been included in the GitHub CI/CD. As a result, there are new vulnerable dependencies when running `yarn audit` on the current repository.

7. Sending funds to user-owned addresses can cause spurious warnings

Status: **Resolved**	
Severity: **Informational**	Difficulty: **Low**
Type: Error Reporting	Finding ID: TOB-UNIMOB-007
Target: Uniswap iOS Wallet token send modal	

Description

When sending tokens to an account, the mobile wallet checks to see whether the user has previously interacted with the recipient address. If not, the user is warned that they are interacting with a new address and are advised to double-check that the address was input as desired. In many cases, this is a desirable warning and may prevent the user from accidentally sending funds to an incorrect address or on the wrong chain.

However, when a user creates a new account in their wallet and then attempts to transfer from an existing account to the new address, they are presented with and must dismiss two consecutive warnings, shown in figure 7.1.

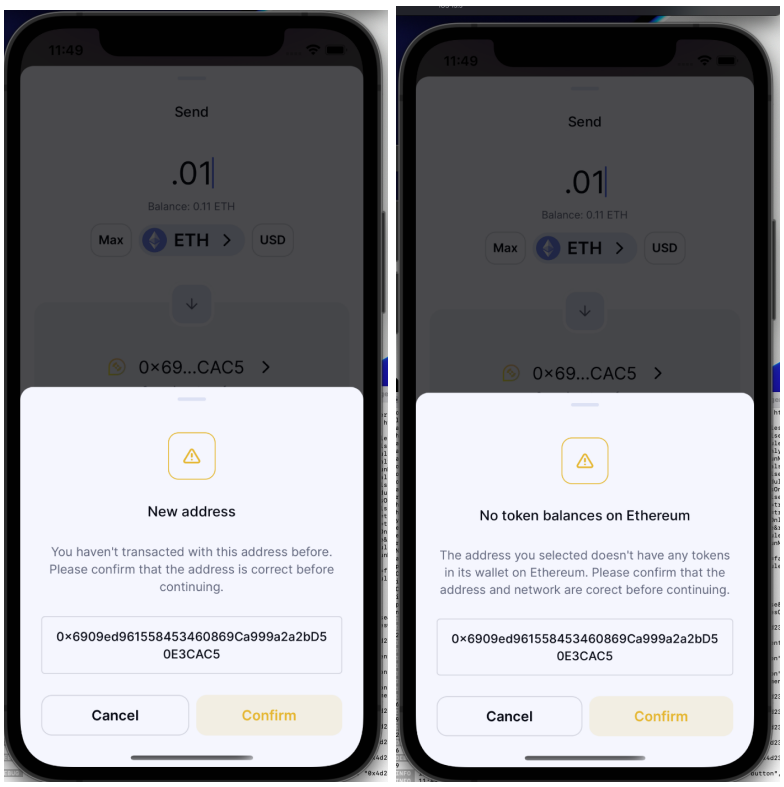

Figure 7.1: New address warning modals

Sending funds to an account for which the user holds the private key is generally safe. Presenting the user with many warnings can lead to “alert fatigue” and encourage the user to dismiss future warnings without consideration. Warnings should be presented only when there is a real risk that needs the user’s attention.

Fix Analysis

This issue has been resolved. The implementation has been updated to not show the warning dialog when the recipient wallet has been created by the user of the mobile application.

8. WalletConnect v1 reuses cryptographic keys for multiple primitives

Status: **Unresolved**	
Severity: **Informational**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-UNIMOB-008
Target: WalletConnect v1 Protocol Specification	

Description

The WalletConnect v1 Specification requires that clients communicate using AES-CBC for encryption along with HMAC-SHA256 to authenticate the data. Both the AES-CBC ciphertext and the HMAC-SHA256 tag are computed using a single shared symmetric key. Although there are no known negative interactions between AES-CBC and HMAC-SHA256, it is best practice to not use the same key for two different cryptographic primitives.

This vulnerability affects the WalletConnect v1 specification, not the Uniswap mobile client usage of the protocol.

Fix Analysis

This issue has not been resolved. However, because the issue is present in the upstream WalletConnect library, there is no way for Uniswap to address it; remediation must be handled by WalletConnect itself.

9. Credentials checked into source control

Status: **Partially Resolved**	
Severity: **High**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-UNIMOB-009
Target: Mobile wallet Git history	

Description

The Uniswap mobile repository has several sensitive credentials checked into source control.

Most notably, the `src/features/wallet/accounts/useTestAccount.ts` file contains a hard-coded mnemonic that controls real world value across several chains; leaking this mnemonic would result in the loss of funds.

```
const MNEMONIC_TEST_ONLY = '[redacted]'
```

Figure 9.1: Mnemonic checked into source of `src/features/wallet/accounts/useTestAccount.ts#8`

Additionally, Covalent, Uniswap, Infura, OpenSea, and Zerion API keys are present in the project's `.env` file. These API keys are used to secure access to more relaxed app-wide rate-limits. If these credentials were leaked, an attacker could exhaust the app's resource allocation, leading to a DoS for all users.

```
COVALENT_API_KEY=ckey_524f68db6e0e43788ba7849da43
COINGECKO_API_URL=https://api.coingecko.com/api/v3/
DEBUG=true
UNISWAP_API_URL=https://dsn76qqamreobir5plkvauf4dm.appsync-api.us-east-2.amazonaws.c
om/graphql
UNISWAP_API_KEY=da2-n65b7e42gzgxzbye4f366ukss4
INFURA_PROJECT_ID=c92fab7a5b7841ecb65f26517b129364
LOG_BUFFER_SIZE=100
ONESIGNAL_APP_ID=5b27c29c-281e-4cc4-8659-a351d97088b0
OPENSEA_API_KEY=d0a4ff8d922e41e29454b86e0426d0f6
SENTRY_DSN=https://a216a11da7354acc9504a688813ff0bf@o1037921.ingest.sentry.io/600606
1
VERSION=0.0.1
ZERION_API_KEY=Demo.ukEVQp6L5vfgxcz4sBke7XvS873GMYHy
```

Figure 9.2: Various API keys checked into source of `.env#6-17`

Lastly, there is a hard-coded Alchemy API key in the project's `hardhat.config.js` file. Although this API key is not used in the app, its exposure could disrupt the development and deployment processes.

```
const mainnetFork = {
  url: 'https://eth-mainnet.alchemyapi.io/v2/lhVWQ3rY2i5_OZtYkU4Lzg_OsDT97Eoz',
  blockNumber: 13582625,
}
```

Figure 9.3: Alchemy API key checked into source of `hardhat.config.js#1-4`

If attackers gain access to the source code of the application, they will have access to these secrets. Additionally, all employees and contractors with access to the repository have access to the above secrets. These secrets should never be kept in plaintext in source code repositories, as they can become valuable tools to attackers if the repository is compromised.

Fix Analysis

This issue is partially resolved. A commit has been made that removes the credentials shown in Figure 9.2. However, the same mnemonic from Figure 9.1 and the Alchemy API key from Figure 9.3 are still present in the latest commit of the repository. Furthermore, the mnemonic still contains real world assets.

Consider rotating this mnemonic and all of the API keys for fresh ones that are never committed to the repository. This GitHub page contains guidance for clearing credentials and other confidential information from a git repository's history.

10. NFTs with SVG images are rendered as HTML

Status: **Unresolved**	
Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-UNIMOB-010
Target: `src/components/images/WebSvgUri.tsx`	

Description

The code in charge of displaying NFTs that contain an SVG image does not sanitize the vector image contents, and renders the image by embedding the image contents in an HTML document. An attacker can serve arbitrary HTML on the URI associated with the token and cause the wallet to render it on a WebView.

```
const getHTML = (svgContent: string) => `
<html>
  <head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, user-scalable=0, shrink-to-fit=no">
  <style>
      <!-- snip -->
    </style>
  </head>
  <body>
    ${svgContent}
  </body>
</html>
`
```

Figure 10.1: The SVG document is embedded in an HTML document without sanitization (`src/components/images/WebSvgUri.tsx#10–40`)

The WebView instance used to render SVG images disables JavaScript, so running arbitrary code is not possible. However, it also allows arbitrary origins to load, so it is possible to perform requests to external hosts (e.g., via an `iframe` tag) or navigate to other external sites (e.g., via a `meta` redirect tag).

```
<WebView
  scalesPageToFit
  javaScriptEnabled={false}
  originWhitelist={['*']}
  scrollEnabled={false}
```

```
  showsHorizontalScrollIndicator={false}
  showsVerticalScrollIndicator={false}
  source={{ html }}
  style={[
    webviewStyle.fullWidth,
    {
      aspectRatio,
      maxHeight,
    },
  ]}
  useWebKit={false}
/>
```

Figure 10.2: JavaScript is disabled, but a wildcard origin is allow-listed (`src/components/images/WebSvgUri.tsx#83–99`)

Fix Analysis

This issue has not been resolved. The implementation has been updated to disable `pointerEvents` on the WebView. However, Uniswap explained the following: “In general we need to use a webview and we cannot sanitize the NFTs because many non-malicious NFTs use different SVG tricks to render properly. We disabled JS and disabled clicking on the webview.”

11. Application does not exclude keychain items from iCloud and iTunes backups

Status: **Resolved**	
Severity: **High**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-UNIMOB-011
Target: `ios/RNEthersRS.swift`	

Description

The Uniswap mobile wallet does not prohibit its keychain items from being saved to an iTunes backup or uploaded to iCloud. Both Apple, Inc. and any attacker with access to a user's iTunes or iCloud backup will have access to that user's private data. Some of the private data stored in the keychain includes mnemonic phrases and private keys for accounts. Figure 11.1 gives an example use of `keychain.set` without the `withAccess` parameter. When omitted, the accessibility class defaults to `accessibleWhenUnlocked`. A second instance can be found on line 109 of the same file.

```
func storeNewPrivateKey(address: String, privateKey: String) {
  let newKey = keychainKeyForPrivateKey(address: address);
  keychain.set(privateKey, forKey: newKey)
}
```

Figure 11.1: `ios/RNEthersRS.swift#L141-L144`

Fix Analysis

This issue has been resolved. The implementation has been updated to include the parameter `accessibleWhenUnlockedThisDeviceOnly`. This prevents keychain data from being stored with a device backup.

12. ShakeBugs may leak mnemonic

Status: **Partially Resolved**	
Severity: **High**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-UNIMOB-012

Target:
- `src/screens/Import/SeedPhraseInputScreen`,
- `src/screens/Onboarding/ManualBackupScreen`,
- `src/screens/SettingsViewSeedPhraseScreen`,
- `src/screens/SettingsManualBackup`

Description

The Uniswap mobile wallet uses ShakeBugs to help with reporting of issues in the app. When a user reports an issue, ShakeBugs can take a screenshot and send it along with the reported issue. If this happens in a screen that shows a mnemonic, the user's mnemonic would be logged to ShakeBugs.

The ShakeBugs documentation provides information on how to mark certain views as "private," which redacts the private information from the screenshot.

Fix Analysis

This issue has been partially resolved. The screen that displays the mnemonic has been updated according to the ShakeBugs recommendations so that it does not include the mnemonic view when making a screenshot. However, Uniswap was not able to get this to work on the screen where a mnemonic needs to be input. As a workaround, Uniswap mentioned a user could manually remove the entered mnemonic before using ShakeBugs to make a screenshot of the input screen.

13. Use of improperly pinned GitHub Actions in Testflight build

Status: **Resolved**	
Severity: **High**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-UNIMOB-013
Target: Mobile wallet GitHub actions	

Description

The GitHub Actions workflows for creating an iOS application build uses several third-party actions that are pinned to a tag or branch name instead of a full commit SHA. This configuration enables repository owners to silently modify the actions. A malicious actor could use this ability to tamper with an application release or leak secrets such as application signing keys.

```
  78    - name: Pod Install
  79      uses: nick-fields/retry@v2
  80      with:
  81        timeout_minutes: 20
  82        retry_wait_seconds: 2
  83        max_attempts: 3
  84        command: cd ios && pod install && cd ..
  85
  86    - name: Build and ship iOS App
  87      run: |
  88        export PATH="/usr/lib/ccache:/usr/local/opt/ccache/libexec:$PATH"
  89        export
CCACHE_SLOPPINESS=clang_index_store,file_stat_matches,include_file_ctime,include_fil
e_mtime,ivfsoverlay,pch_defines,modules,system_headers,time_macros
  90        export CCACHE_FILECLONE=true
  91        export CCACHE_DEPEND=true
  92        export CCACHE_INODECACHE=true
  93        ccache -s
  94        set -o pipefail
  95        yarn deploy:ios:alpha
  96        ccache -s
  97      shell: bash
  98      env:
  99        APP_IDENTIFIER: ${{ secrets.APP_IDENTIFIER }}
 100        APPLE_ID: ${{ secrets.APPLE_ID }}
 101        APPLE_APP_ID: ${{ secrets.APPLE_APP_ID }}
 102        APPLE_TEAM_ID: ${{ secrets.APPLE_TEAM_ID }}
 103        URL_TO_FASTLANE_CERTIFICATES_REPO: ${{
secrets.URL_TO_FASTLANE_CERTIFICATES_REPO }}
```

```
 104         MATCH_PASSWORD: ${{ secrets.MATCH_PASSWORD }}
 105         FASTLANE_APPLE_APPLICATION_SPECIFIC_PASSWORD: ${{
secrets.FASTLANE_APPLE_APPLICATION_SPECIFIC_PASSWORD }}
 106         CI: true
 107         CI_KEYCHAIN_NAME: 'CI_KEYCHAIN'
 108         CI_KEYCHAIN_PASSWORD: ${{ secrets.CI_KEYCHAIN_PASSWORD }}
 109         GIT_BRANCH_NAME: ${{ github.ref }}
 110         GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Figure 13.1: The `nick-fields/retry` action is pinned only to a tag and can access the GitHub token among other secrets (`.github/workflows/fastlane.yml#L78-L110`)

Fix Analysis

This issue has been resolved. All external GitHub actions are now pinned using a commit SHA.

A. Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

B. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.