Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Audit-Report SolidiFi Wallet Staking Feature 01.2024

Cure53, Dr.-Ing. M. Heiderich, MSc. H. Moesl-Canaval, MSc. A. Schloegl

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

“DeFi, Trading & Payments. All in one. At SolidiFi we specialize in blockchain applications specifically for the XRPL, XinFin and Flare blockchains.”

From https://solidifi.app/

This report has been compiled to elucidate the outcomes of a Cure53 cryptography review and source code audit against the SolidiFi Wallet application’s staking feature, which was performed in early 2024 and funded by CloudElements.

To offer some background information, the assessment was requested during initial discussions held in November 2023. Cure53 then assembled a three-person team of highly proficient pentesters and scheduled the review for CW02 January 2024. In total, the client invested four working days to achieve the expected degree of coverage against the targets. These were placed into a single Work Package (WP) entitled *WP1: Cryptography reviews & code audits against SolidiFi staking feature*.

In actuality, the SolidiFi Wallet application has been examined by Cure53 previously, specifically during the audit performed in January and February 2023 (documented under report CLE-01). Albeit, one must mention that this current project focused on a specific feature rather than the entire application, in contrast to the previous engagement.

To assist the undertakings, sources, test-supporting documentation, and other assorted pieces of information or data required for access or general purposes were handed over in advance. The provision of sources rendered the evaluation process compliant with a white-box methodology.

A number of preparatory initiatives were completed ahead of the actual reviews. These were performed in late December 2023 (CW51) and early January 2024 (CW01) to create an ideal and hindrance-free testing environment.

A dedicated and shared Slack channel was established to foster communications between the two organizations. All employees from both sides that played an active part in this project were invited to join the channel. Cure53 would like to extend its appreciation to CloudElements for a seamless collaboration process, which required minimal cross-team queries due to the excellent scope composition. The test team relayed a swathe of status updates when required, elaborating both the general audit progress and noteworthy findings, though live reporting was deemed surplus to requirements.

Concerning the security impacting discoveries, a total of five tickets were documented following reasonable examination depth against the WP1 item in focus. Only one of those was categorized as a security vulnerability, while the remaining four pertained to best practice advice or minor detriments.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Cure53 considers this a small yield of findings in general; perhaps expected, considering the concise nature of the scope and audit time frame. Nonetheless, similar procedures against alternative products have generated a substantially higher volume of faults than those witnessed here, indicating robust security paradigms.

The altogether positive outcome is compounded by the fact that no new security vulnerabilities were identified at all. However, several findings noted in CLE-01 (including one vulnerability) have either not yet been mitigated or require additional amendments to the developer team's fixes. Cure53 would be remiss not to restate the importance of correctly and swiftly addressing all findings, regardless of their perceived exploitation probability or severity score.

To summarize, Cure53 is pleased to confirm that the SolidiFi Wallet staking feature exhibits sufficient stability under the current configuration. The guidance proposed throughout this document should nonetheless be adhered to in order to fortify the premise to an industry leading standard.

Onward, the report continues by detailing the scope and test setup, itemizing the software components, methodology, and materials, inter alia.

Following, a dedicated section presents the exhaustive test methodology, demonstrating the areas covered and tests executed. This transparency hopefully verifies the meticulousness of the analyses, despite the limited number of identified findings.

Subsequently, the report systematically lists all findings, starting with the detected vulnerabilities and followed by the general weaknesses. Each finding will be accompanied by a detailed technical explanation, a Proof-of-Concept (PoC) or steps to reproduce where applicable, code excerpts, and specific mitigation or remediation recommendations.

Finally, the report concludes with a comprehensive summary of Cure53's overall impressions and insights. This section offers a critical estimation of the staking feature's perceived security posture, highlighting demonstrable strengths as well as areas for improvement.

Fix note: *This report has been modified to include fix notes for each issue that has been successfully closed by the maintainers. The rest of the report was left untouched.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Cryptography reviews & source code audits against SolidiFi Wallet staking feature**
  - **WP1**: Cryptography reviews & code audits against SolidiFi staking feature
    - **Source code:**
      - **Branch:**
        - master
      - **Commit hash:**
        - 9a31a9f
    - **Documentation:**
      - **Visual overview of staking within SolidiFi:**
        - https://www.youtube.com/watch?v=UL8j_4-l-y4
      - **Technical overview of a tool developed by Flare:**
        - This tool uses a similar method, including the usage of FlareJS
          - https://docs.flare.network/user/staking/staking-cli/
    - **Staking documentation:**
      - *Solidifi-staking-documentation.pdf*
    - **SolidiFi Wallet documentation:**
      - *Solidifi-wallet-design-documentaion-v13.pdf*
    - **Flare Wallet:**
      - **Sample wallet address:**
        - 0x32b5B06815Ecc2F74446361078BA08ab398d66B8
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Test Methodology

This section documents the testing methodology applied by Cure53 during this CLE-02 project and discusses the resulting coverage, shedding light on how various components were examined. Further clarification concerning areas of investigation subjected to deep-dive assessment is offered, especially in the absence of significant security vulnerabilities detected.

Accordingly, Cure53 hopes that the following passages enhance understanding of the methods and techniques employed by the testers during the assessment procedures. The primary focus was to evaluate SolidiFi's security posture through validation of both successful and unsuccessful approaches, offering insight into amelioration opportunities:

- To initiate proceedings, the review team analyzed the staking feature and all associated classes located in *src/screens/Staking*. These undertakings confirmed that sensitive data handling and transaction signing are conducted using the ethers[1] library. The avoidance of custom cryptographic operations means that the implementation adheres to industry best practices. This vetting process also encompassed an examination of the JSON-RPC calls utilized within the business logic, which ultimately verified that this area was risk averse.

- Regarding the detected flaws, the Cure53 consultants immediately verified the ability to install the app's Play Store version on multiple rooted and emulated devices, as filed under ticket CLE-02-001. As a consequence, one could leverage the app for dynamic testing via the Frida framework[2]. The staking flow entails a number of activities, including (but not limited to) the transfer of funds between the c- and p-chains, c- and p-chain address binding, reward claims, and the actual staking transactions. Here, the technical team was able to verify and intercept all internally initiated JSON-RPC calls. However, despite strenuous efforts, Cure53 was unable to identify a breach strategy that would permit private key extraction or fraudulent signature generation on a device that is not attacker-controllable.

- The assessors utilized the broken root detection to perform supplementary investigations of the SolidiFi app and intercept TLS traffic from the mobile device using corresponding proxy software (i.e., Burp Suite[3]). Here, Cure53 was able to successfully intercept the API key employed for the Flare explorer API, as demonstrated in ticket CLE-02-002. In spite of the SolidiFi team's proactive measures to address the initial vulnerability during the assessment, this report underscores an additional attack vector that facilitates API exfiltration, which must be monitored accordingly.

[1] https://docs.ethers.org/v5/getting-started/
[2] https://frida.re/
[3] https://portswigger.net/burp

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Lastly, Cure53 scrutinized the previous penetration test report to ascertain whether follow-up remediation actions had been implemented by the CloudElements developers. As such, the following pertinent observations were made:

- The PIN is now stored through the adoption of Expo SecureStore[4], which assures the confidentiality and integrity of data at rest using AES-GCM.

- The original recommendation to modify the launch mode for *com.cloudelements.solidifi.MainAcitivity* has not been complied with, as reported in ticket CLE-02-004.

- The internal browser now only operates on *https://* connections, while traffic between the WebView and React Native code is encrypted using AES-GCM.

- Blurring is now correctly implemented in the app, nullifying any leakage potential when the app assumes a backgrounded state. Additionally, all attempts to mirror the device's display to a computer using *scrcpy*[5] were met with a black screen.

- Following further testing, Cure53 verified that Android backups are no longer permitted.

- URL parsing is now handled by the URL class, while protocol verification is performed using the protocol property.

- Root detection has been incorporated via jail-monkey[6]; however, the ability to install and operate the app on two alternate rooted devices and an emulator confirmed that these checks are wholly insufficient. For supporting guidance, please refer to ticket CLE-02-001.

[4] https://www.npmjs.com/package/expo-secure-store
[5] https://github.com/Genymobile/scrcpy
[6] https://www.npmjs.com/package/jail-monkey

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *CLE-02-001*) to facilitate any future follow-up correspondence.

CLE-02-004 WP1: Unfixed vulnerability from previous audit *(Medium)*

***Fix note**: This issue was fixed and the fixes were verified by Cure53, the problem as described no longer exists.*

This ticket serves to reiterate a specific vulnerability identified during the previous source code audit that remains unmitigated. An analysis of both the source code and application provided for this test iteration confirmed that the following flaw remains outstanding and requires remediation at the earliest opportunity.

CLE-01-003 WP1: Android application vulnerable to task hijacking

Generally speaking, a plethora of internal tasks function collectively and simultaneously within any Android application. Whenever the application operates, a back stack is actioned that defines the previous task opened or achieved. In this context, Cure53's endeavors indicated that the application is still vulnerable to task hijacking. Accordingly, attackers can manipulate victim users into unintentionally opening a malicious application that deviates from the originally intended entity.

To mitigate this issue, Cure53 recommends setting the launch mode to *singleInstance*, as reported during the prior CLE-01 engagement. This alteration will prevent other activities from joining tasks belonging to the application. Alternatively, the CloudElements developers could incorporate a quick and effective fix by configuring *taskAffinity=""*.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

CLE-02-001 WP1: Insufficient root and emulator checks for Android devices *(Low)*

***Fix note**: This issue was fixed and the fixes were verified by Cure53, the problem as described no longer exists.*

Cure53's dynamic testing of the SolidiFi Android application verified the ability to launch the app successfully on both rooted and emulated devices, contrary to design goal 7.5.1 outlined in the accompanying PDF document. The first device employed for this procedure represented a OnePlus 8 operating LineageOS, rooted with Magisk. In this instance, Magisk utilized its Zygisk feature, which operates a portion of Magisk within the Zygote daemon, thus concealing its presence from specific apps. However, Magisk was not actively obfuscated from the SolidiFi app. Notably, this device failed to pass any of the Play Integrity API[7] checks.

Moreover, Cure53 extended this evaluation process by using a second device, i.e., a Pixel 4, which was also rooted with Magisk. On this occasion, measures to obscure the rooting were not initiated, indicating that the SolidiFi app's current root detection functionalities are suboptimal.

Despite the fact that the library employed for root detection accurately identifies the device in question as rooted, the steps outlined below confirm that the app fails to implement any measures to restrict operations on a device of this nature:

Steps to reproduce:

1. Install and run the Frida[8] server on the device, then connect it to a PC.
2. Initiate SolidiFi locally with the following PoC script command:

   Command:
   ```
   frida -U -l root_check_poc.js -f com.cloudelements.solidifi
   ```

3. Save the following PoC script inside *root_check_poc.js* within your current operational directory:

   ```
   var Main = function() {
     Java.perform(function() {
   ```

[7] https://developer.android.com/google/play/integrity
[8] https://frida.re/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
Java.use('com.gantix.JailMonkey.Rooted.RootedCheck').isJailBroken.imp
lementation = function() {
      let retval = this.isJailBroken();
      console.log(`isJailBroken returned: ${retval}`)
      return retval
    }
  });
};

Java.perform(Main);
```

4. Run the aforementioned script and note that the return value of JailMonkey's *isJailBroken* function is logged.

   Example output:
   `isJailBroken returned: true`

To mitigate this issue, Cure53 advises performing a thorough examination of the application's integrity check to identify and rectify the logic error that overlooks the failed root check outcome. According to the Frida script output, the detection mechanism appears to function correctly. However, the issue emanates from the method by which this information is processed and likely originates from either the JailMonkey library or SolidiFi codebase. As such, one can safely assume that the subpar emulator detection protocols suffer from a correlating (if not identical) weakness.

CLE-02-002 WP1: Flare Portal API key leakage *(Medium)*

***Fix note**: This issue was fixed and the fixes were verified by Cure53, the problem as described no longer exists.*

During the course of the security review, Cure53 found that the application inadvertently revealed Flare Portal API keys, potentially granting unauthorized parties access to paid API features. In addition, the observation was made that the API key is extractable from the Android application APK file by utilizing tools such as *hermes-dec*[9].

This unauthorized access may incur significant financial charges and could result in the enforcement of limitations against the application's service usage. The paid Flare Portal API solution facilitates a gateway to Flare's blockchains and connected chains. With this in mind, API key leakage might lead to numerous plausible ramifications, such as monthly quota exhaustion by automated interactions; extensive financial damage in the event that billing limits are exceeded; and functional restrictions upon the application due to Denial-of-Service (DoS) attacks.

[9] https://github.com/P1sec/hermes-dec

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The following API key was located within the app during this test iteration:

API key identified:
`TzA77Y<redacted>`

This API key enables permission to various Flare API endpoints, while the subscription incurs $0.0002 per API call once the subscription limit is reached. The following cURL command can be adopted to misuse the leaked API key:

cURL:
```
$ curl -X GET "https://flare-explorer.flare.network/api?
module=account&action=eth_get_balance&address=0x32b5B06815Ecc2F74446361078B
A08ab398d66B8&block=latest" -H "accept: application/json" -H "X-Apikey:
TzA77Y<redacted>"
```

To mitigate this issue, Cure53 advises implementing an orchestration layer (serverless function) between the app and resource that can forward the request with the required API key or secret. This revised approach would prevent direct API consumer access to the secrets in question. Furthermore, the developer team should ensure that only mobile-related *User-Agents* are able to submit data to the serverless function.

Pertinently, the issue leading to the accidental initial disclosure of the API key in a request has already been fixed by the developer team during the active assessment phase. However, the shortcoming related to the extraction of API keys from files present on the device requires resolution by means of the aforementioned orchestration layer or a similar strategy.

CLE-02-003 WP1: Unfixed miscellaneous issue from previous audit *(Info)*

***Fix note**: This issue was fixed and the fixes were verified by Cure53, the problem as described no longer exists.*

This ticket serves to pinpoint a specific miscellaneous issue detected during the previous source code audit that remains unresolved. Scrutiny of both the source code and the application provided for this project confirmed that the following issue is outstanding and requires attention.

CLE-01-002 WP1+2: BugSnag API key hardcoded in config

Decompiling the mobile application revealed that the BugSnag API key was still insecurely baked into the *AndroidManifest.xml* and *Info.plist* files. Consequently, adversaries can abuse this key to send fake reports or source maps to the platform dashboard, thus compromising BugSnag data integrity.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 advises introducing an orchestration layer (serverless function) between the app and resource that can forward the request with the required API key or secret, as proposed in the preceding CLE-02-002 ticket.

CLE-02-005 WP1: Vulnerable dependencies *(Low)*

***Fix note**: This issue was fixed and the fixes were verified by Cure53, the problem as described no longer exists.*

During the security assessment, the observation was made that the SolidiFi app leveraged outdated versions that are vulnerable to certain security risks. The following software packages were identified as out-of-date and potentially insecure. Notably, the version information provided is based on data collected at the time of testing. However, the exploitation likelihood associated with these vulnerabilities depends on how the relevant functionality is currently used in the targeted application.

The vulnerable dependencies can be obtained by running the following command within the source code folder:

Command:

```
$ npm audit

follow-redirects  <1.15.4
Severity: high
Follow Redirects improperly handles URLs in the url.parse() function -
https://github.com/advisories/GHSA-jchw-25xp-jwwc
fix available via `npm audit fix`
node_modules/follow-redirects

1 high severity vulnerability
```

Due to the limited scope and time allocation, the full potential impact of the findings remains undetermined. Further internal investigations are recommended to assess this aspect comprehensively.

The implementation of robust supply chain security is oftentimes significantly challenging. There are often no simple or universal solutions, while the effectiveness of the chosen frameworks can be heavily dependent on specific library versions.

To mitigate this issue, Cure53 recommends upgrading all affected libraries and establishing a policy to ensure libraries remain up-to-date moving forward. This will allow the framework to benefit from patches that have been rolled out for all flaws that have been previously detected across various solutions. To achieve this, the CloudElements team could leverage NPM's *npm audit fix* functionality. However, the degree of protection may vary. Similarly,

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

keeping all dependencies updated will inevitably become increasingly difficult to achieve if the framework leverages an extensive number of third-party libraries.

Under certain circumstances, one may have to resort to either sending Pull Requests (PRs) to the library maintainer or even forking the library entirely. CloudElements could consider assigning a developer as the task owner to ensure this issue is not neglected or deprioritized. Lastly, certain libraries may need to be replaced with actively maintained alternatives in the long term.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

This feature audit procedure represents Cure53's second evaluation of the SolidiFi scope following the inaugural 2023 iteration. CLE-01 was exclusively dedicated to the Android and iOS applications and was approached via a mobile-specific penetration testing perspective, while specific features were omitted from holistic scrutiny. In contrast, CLE-02 primarily focused on newly introduced features, prioritizing the app's staking feature for heightened research.

Cure53 maintained consistent communication with the client via the private Slack channel formed for this purpose. The channel was conducive to an effective analysis and the maintainers readily provided assistance whenever requested.

The source code was accompanied by comprehensive documentation that meticulously detailed all design objectives, use cases, and security considerations, allowing the Cure53 team members to quickly familiarize themselves with the underlying infrastructure. The codebase, written in Typescript, boasts optimized structuring and clear organization, with the source code presented in an easily comprehensible manner. This clarity eased the technical team's understanding of the corresponding functionalities.

Firstly, to ascertain the propensity for client-side issues such as DOM-based XSS, Cure53 perused the connected client-side JavaScript code. This verified that the React framework has been correctly employed and that the implementation avoids associated pitfalls. The management of sensitive data related to staking and cryptocurrency transactions is conducted using the ethers library, which is an astute and widely adopted solution. Moreover, critical assets such as private keys that require consistent protection are securely encrypted and stored using the Expo SecureStore module.

Unfortunately, the ineffective jailbreak and emulator detection allowed the testers to dynamically analyze the Android app, scrutinize HTTP traffic, and retrieve API keys. These malicious activities have been documented in two distinct tickets, CLE-02-001 and CLE-02-002. Additionally, one must highlight that the API key is directly extractable from the Android APK file statically through reverse engineering.

Despite the core focus on staking (and thus API calls to backend systems), none of the identified issues permitted the exfiltration or leakage of critical user assets. Furthermore, the staking logic blocked all attempts to generate fraudulent signatures or instigate integrity attacks.

The SolidiFi app relies on third-party libraries for specific operations, which means that SolidiFi's security efficacy is intrinsically tied to the resilience and defensive capabilities of said libraries. In light of this, Cure53 strongly recommends enforcing a rigorous update regimen for all dependencies within the staking feature.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

This proactive approach is crucial for timely identification and remediation of potential security vulnerabilities within third-party libraries. While one outdated library was discovered during testing, as documented in ticket CLE-02-005, the project supervisors's prompt resolution demonstrates admirable commitment to maintaining a secure software environment.

The majority of the drawbacks detected during the previous audit have been resolved. However, some issues still persist or have not been comprehensively mitigated, as filed under tickets CLE-02-004 and CLE-02-003.

To conclude, Cure53 collated substantial evidence indicating that top-tier security performance was a key consideration during initial system development. The CloudElements team's conformity to wider best practices is irrefutable; however, all points of contention outlined in this report should be heeded to elevate the platform's cyber confidence even further.

The commitment to long-term security through regular assessments of the SolidiFi mobile apps should be commended and continued. Given the inherent risks associated with mobile applications handling sensitive data, particularly within the financial domain, maintaining a robust security posture is not only recommended but critical. Any prevalent vulnerabilities, especially within the staking feature, could result in significant financial losses and reputational damage. Therefore, prioritizing ongoing security appraisals with a focus on identifying and mitigating potential threats remains paramount.

Cure53 would like to thank Ferdi Zoet from the CloudElements team for his excellent project coordination, support, and assistance, both before and during this assignment.