QuillAudits

AUDIT REPORT

January 2025

For

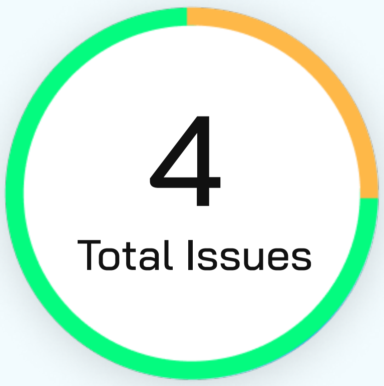
safle

Table of Content

Executive Summary

Project Name	Safle
Overview	Safle Vault is a non-custodial, flexible and highly available crypto wallet which can be used to access dapps, send/receive crypto and store identity. Vault SDK is used to manage the vault and provide methods to generate vault, add new accounts, update the state and also enable the user to perform several vault related operations.
Audit Scope	The scope of this pentest was to analyze the Source code for quality, security, and correctness.
Contracts in Scope	https://github.com/getsafle/safle-vault Version:2.9.3 Branch: main (commit:408ff84d6da2a61dd8f41b734dc9c8889a4c9c10)
Review 1	7th January 2025 - 22nd January 2025
Updated Code Received	13th January 2025
Final Review	24th January 2025
Fixed In	version 2.9.4. https://github.com/getsafle/safle-vault

Number of Issues per Severity



🟥	High	0 (0%)
🟧	Medium	1 (25%)
🟩	Low	3 (75%)
🟪	Informational	0 (0%)

Severity

Issues

	🟥 High	🟧 Medium	🟩 Low	🟪 Informational
Open	0	0	0	0
Resolved	0	**1**	**3**	0
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Improper Authentication
- ✓ Improper Resource Usage
- ✓ Improper Authorization
- ✓ Insecure File Uploads
- ✓ Insecure Direct Object References
- ✓ Client-Side Validation Issues
- ✓ Rate Limit
- ✓ Input Validation
- ✓ Injection Attacks
- ✓ Cross-Site Scripting (XSS)
- ✓ Cross-Site Request Forgery
- ✓ Security Misconfiguration
- ✓ Broken Access Controls
- ✓ Insecure Cryptographic Storage
- ✓ Insufficient Cryptography
- ✓ Insufficient Session Expiration
- ✓ Insufficient Transport Layer Protection
- ✓ Unvalidated Redirects and Forwards
- ✓ Information Leakage
- ✓ Broken Authentication and Session Management
- ✓ Denial of Service (DoS) Attacks
- ✓ Malware
- ✓ Third-Party Components

And More..

Techniques and Methods

Throughout the pentest of application, care was taken to ensure:

- Information gathering – Using OSINT tools information concerning the web architecture, information leakage, web service integration, and gathering other associated information related to web server & web services.
- Using Automated tools approach for Pentest like Nessus, Acunetix etc.
- Platform testing and configuration
- Error handling and data validation testing
- Encryption-related protection testing

Tools and Platforms used for Pentest:

Burp Suite	Acunetix	Nmap
DNSenum	Neucli	Metasploit
Dirbuster	Nabbu	Horusec
SQLMap	Turbo Intruder	Postman
Netcat	Nessus	And Many more..

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

🟥 High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

🟧 Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

🟩 Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

🟪 Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

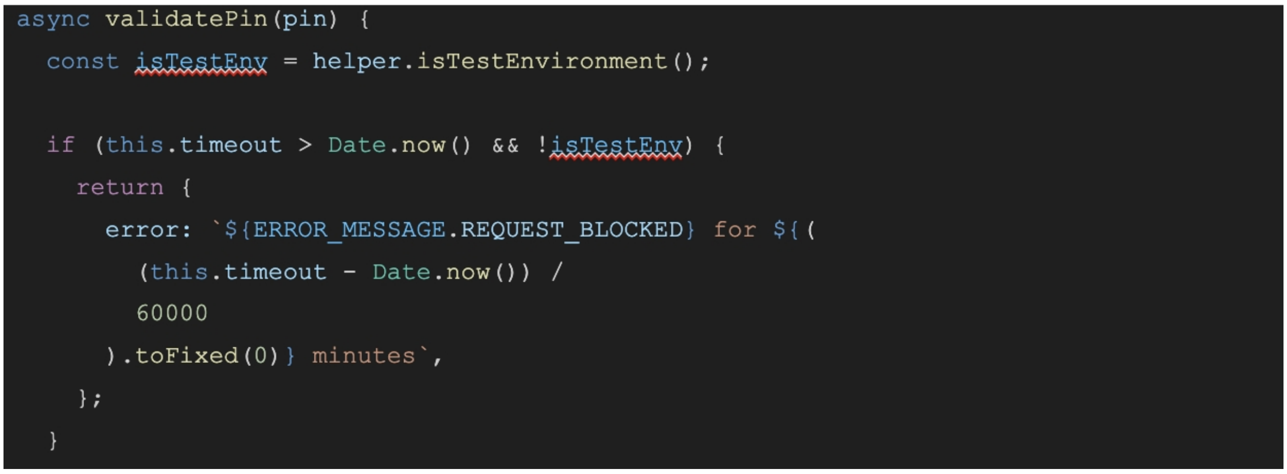
Medium Severity Issues

1. Timeout not updated

Resolved

Description

The code sets the value of timeout to 0 on line 25 in the code. Here in the following code snippet -



```
async validatePin(pin) {
  const isTestEnv = helper.isTestEnvironment();

  if (this.timeout > Date.now() && !isTestEnv) {
    return {
      error: `${ERROR_MESSAGE.REQUEST_BLOCKED} for ${(
        (this.timeout - Date.now()) /
        60000
      ).toFixed(0)} minutes`,
    };
  }
```

the application directly compares the value of timeout and current date. The problem is the timeout variable is set to 0 and is not updated to current timeout which means this condition will always be false. 0 is never greater than the current date in timestamp format and hence the if condition will always stay false.

Vulnerable Endpoint

https://github.com/getsafle/safle-vault/blob/main/src/lib/keyring.js#L66

Impact

The condition is pre-defined false irrespective of what input comes from the user making the condition inefficient.

Recommendation

Either Update the value of timeout variable before comparing the value of timeout and current date. Or else remove this condition as it is always going to be invalid.

Low Severity Issues

2. Weak Cryptography in vault.js

Resolved

Description

The encrypt and decrypt functions in initializeKeyringController use CryptoJS.AES without salting or strong key derivation.

Vulnerable Location

https://github.com/getsafle/safle-vault/blob/main/src/lib/vault.js#L48

Recommended Fix

Use key derivation algorithms with strong salting mechanisms.
Example:

```
const bcrypt = require('bcrypt');
const salt = bcrypt.genSaltSync(10);
const key = bcrypt.hashSync(password, salt);
```

Impact

Sensitive data (e.g., mnemonics, encryption keys) can be decrypted if an attacker gains access to the ciphertext.

3. Unsecured Sensitive Parameter Handling

Resolved

Description

Sensitive parameters like mnemonic, encryptionKey, and pin are handled insecurely, passed directly in memory, and potentially logged. This exposes them to unauthorized access.

Vulnerable Code

https://github.com/getsafle/safle-vault/blob/main/src/lib/vault.js#L229

Recommendation

Minimize exposure by using secure storage and clearing sensitive data after use.

```
function securelyHandleMnemonic(mnemonic) {
   try {
      console.log('Processing mnemonic...'); // Do not log sensitive data.
   } finally {
      mnemonic = null; // Clear mnemonic from memory.
   }
}
```

4. Insecure Rate Limit

Resolved

Description

The application only uses the “limiter” library to handle rate limits. However, this might not be enough. Additional IP based restrictions should be implemented to restrict attacks.

Vulnerable File

src/lib/keyring.js:7

Impact

If logic of limiter is bypassed by any methods in future, the attacker would be able to bypass rate limits set on pin validation and other methods

Recommendation

Implement IP based restrictions as well to block multiple attempts from malicious IP addresses.

Closing Summary

In this report, we have considered the security of the Safle SDK. We performed our audit according to the procedure described above.

Some issues of High, medium, low, and Informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Pentest security audit provides services to help identify and mitigate potential security risks in Safle SDK. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security ofSafle SDK. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Safle SDK Team to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

QuillAudits

6+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1K+** Projects Secured

Follow Our Journey

AUDIT REPORT

January 2025

For

safle

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillhash.com