



BLOCKCHAIN WALLET AUDIT REPORT

for

HARMONY



Prepared By: Shuxiao Wang

Jan. 03, 2021

Document Properties

Client	Harmony
Title	Blockchain Wallet Audit Report
Target	OneWallet
Version	1.0-rc
Author	Huaguo Shi
Auditors	Huaguo Shi, Xin Li
Reviewed by	Chiachih Wu, Shuxiao Wang
Approved by	Xuxian Jiang
Classification	Confidential

Version Info

Version	Date	Author	Description
1.0-rc	Jan. 03, 2021	Huaguo Shi	Phase I Audit Report

Contact

For more information about this document and its contents, please contact PeckShield Inc. [\[6\]](#) .

Name	Shuxiao Wang
Phone	+86 173 6454 5338
Email	contact@peckshield.com

Contents

1	Introduction	4
1.1	About OneWallet	4
1.2	About PeckShield	5
1.3	Methodology	5
1.4	Disclaimer	9
2	Findings	10
2.1	Summary	10
2.2	Key Findings	11
3	Detailed Results	12
3.1	Vulnerability in Outdated Web Plug-in lodash	12
3.2	Vulnerability in Plug-in vue-styled-components	13
4	Conclusion	14
	References	15

1 | Introduction

Given the opportunity to review the design document and related implementation of OneWallet, we have accordingly audited the client applications under `web` Platform. We outline in this report our systematic method to evaluate potential security issues in current implementation, expose possible semantic inconsistencies between the source code and the design specification, and provide additional suggestions and recommendations for improvement. Our results show that the given implementation of OneWallet can be further improved due to the presence of several issues related to either security or performance. This document describes our audit results in detail.

1.1 About OneWallet

The Harmony [1] blockchain is a high performance blockchain with advanced techniques in secure and random state sharding. Specifically, the Harmony mainnet supports thousands of nodes in multiple shards with the capability of producing blocks in a few seconds with instant finality. The audited OneWallet is a browser extension wallet that supports native DApps running in the Harmony blockchain, and allows for users to smoothly receive, send, store, and exchange different cryptocurrencies.

The basic information of OneWallet is shown in Table 1.1:

Table 1.1: Basic Information of OneWallet

Item	Description
Issuer	Harmony
Website	https://harmony.one
Type	Wallet App
Platform	Web
Coding Language	Type Script
Audit Method	White-box
Latest Audit Report	Jan. 03, 2021

The phase *I* audit focuses on the following code changes in the `onewallet` Git repository:

- <https://github.com/harmony-one/onewallet/commit/89819835641b03e5fa69b3cc01be9db06ef4af11#diff-0f3e1ac14ca42eb8636503427d2d57f955647f9694c68664e608bd68c4b1820aL270-L275>
- <https://github.com/harmony-one/onewallet/commit/34e467d5cc2ab204a1978de597092da5c084cd16#diff-6124443503d55dccdd98f25daefef3ada7b68e22e51a41fe86ff5e8b6fa09a77L172-L174>

1.2 About PeckShield

PeckShield Inc. is a leading blockchain security company with the goal of elevating the security, privacy, and usability of the current blockchain ecosystems by offering top-notch, industry-leading services and products including security audits. We are reachable at Telegram (<https://t.me/peckshield>), Twitter ([twitter](#)), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
		Likelihood		

1.3 Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [5]:

- **Likelihood**: represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- **Impact**: measures the technical loss and business damage of a successful attack;
- **Severity**: demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, and *Low* shown in Table 1.2.

To evaluate the risk, we go through a checklist of items and each is labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, it is considered safe regarding the check item. For any discovered issue, we might further run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. This audit is considered as the phase *I* that covers only those security issues in OneWallet v1.0.6 (<https://github.com/harmony-one/onewallet> (52f056f)). The scheduled phase *II* audit is expected to cover the full concrete list of check items as shown in Table 1.3.

In particular, our audit is performed according to the following audit steps:

- Automated Static Analysis: We first begin the analysis by detecting common code and application-level vulnerabilities with a home-made automated static analysis tool. This tool is helpful to quickly identify known coding bugs, and we will then manually verify (reject or confirm) those issues found by our tool. Specifically, throughout the vulnerability scanning process, we will reproduce each issue based on the error log files generated by the vulnerability analysis tool. For each vulnerability case, we will further analyze the root cause and check if it is indeed a vulnerability. Once a risk is confirmed, we will analyze it further as part of a white-box audit, with a better understanding of the associated business logic and context.
- Business Logic Analysis: We next understand business logics, review system-wide operations, examine the interactions between different components, and place wallet-related logic under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of applications from the perspective of proven programming practices for the targeted platforms.

To better describe each issue we identified, we also categorize the findings based on Common Weakness Enumeration (CWE-699) [4], which is a community-developed list of software weakness types to better classify and organize weaknesses around concepts frequently encountered in software development. We use the CWE categories in Table 1.4 to classify our findings.

Table 1.3: The Full Audit Checklist

Platform	Category	Check Item
Web	Third-party Plug-in	Security of using third-party plugins
	Common Secure Development	Correct Random Number
		Sensitive Information Printed by Log
	Communication Security	Network Communication Security
		Network Proxy Security
	Data Protection	Program Data Arbitrarily Backup
		Global File Security
		Configuration File Security
		Clipboard Security
		Anti-screenshot
	Business Logic	Private Key Generating
		Encryption Algorithm
		Password Strength
		Mnemonic Words Generating
		Private Key and Mnemonic Words Storage
		Local Sensitive Data Storage
		Mnemonic Words Import
		Private Key Import
		User Input Security
		Transaction Logic
		Wallet Communication
		Password Strength
		Password Updating
		Server Interaction Logic
		Functionality Integrity

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logic	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

1.4 Disclaimer



Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of blockchain software. Last but not least, this security audit should not be used as investment advice.



2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the OneWallet implementation. During the phase I of our audit, we limit the scope of the audit to those code changes listed in Section 1.1, the code compilation procedure, and related code logic.

Severity	# of Findings	
Critical	0	
High	1	
Medium	0	
Low	1	
Informational	0	
Total	2	

We have so far identified a list of potential issues, which include issues that can be overlooked from the wallet developer's perspective such as the wallet's runtime system security. Other security issues are mainly related to the unusual interaction between the application and the secure chip. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determine a few issues of varying severities that need to be brought up and paid more attention to, which are categorized in the above table. The detailed discussions of each of them are in the next Section.

2.2 Key Findings

Overall, the audited OneWallet system is well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 high-severity vulnerability and 1 low-severity vulnerability.

Table 2.1: Key Audit Findings

ID	Severity	Platform	Title	Category	Status
PVE-001	High	Web	Vulnerability in Outdated Web Plug-in lodash	Coding Practices	Confirmed
PVE-002	Low	Web	Vulnerability in Plug-in vue-styled-components	Coding Practices	Confirmed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the actual deployment. The risk-control mechanisms should kick in at the very moment when the application or software is deployed. Please refer to Section 3 for details.



3 | Detailed Results

3.1 Vulnerability in Outdated Web Plug-in lodash

- ID: PVE-001
- Severity: High
- Likelihood: Medium
- Impact: High
- Platform: Web
- Target: N/A
- Category: Coding Practices [3]
- CWE subcategory: CWE-1104 [2]

Description

It is a common practice during code development to import a number of third-party or unofficial libraries. Therefore, validating the security of dependent libraries is critical to overall application security. During our analysis, we notice that the imported library `lodash v3.10.1` contains several known vulnerabilities, the most severe one could lead to so-called `prototype pollution` that in essence causes the addition or modification of an existing property to exist on all objects. More details about `prototype pollution` can be found at <https://www.npmjs.com/advisories/1065>.

Moreover, our analysis also uncovers other severe issues in the developer's dependency library (`devDependencies`). An example is the imported library `uglifyjs-webpack-plugin v2.2.0` in `devDependencies`, which has various issues of `cross-site scripting` and `remote code execution`. In the meantime, we clarify that this library is only used internally in the development version, and does not affect the release version.

Recommendation Upgrade the dependent `lodash v3.10.1` to version `v4.17.20` or above. Also, we highly recommend executing `npm audit fix` to check the existence of any 0day vulnerabilities after the version change of the dependent libraries.

Status This issue has been confirmed by the team.

3.2 Vulnerability in Plug-in vue-styled-components

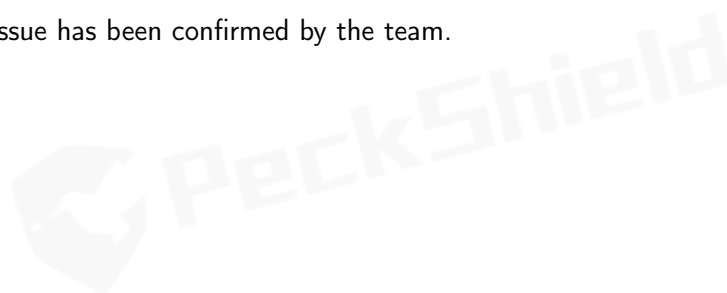
- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Medium
- Platform: Web
- Target: N/A
- Category: Coding Practices [3]
- CWE subcategory: CWE-1104 [2]

Description

As mentioned in Section 3.1, it is a common practice during code development to import a number of third-party or unofficial libraries and validating the security of dependent libraries is critical to overall application security. During our analysis, we also notice that the imported library `vue-styled-components` v1.5.1 used an older version `node-fetch`, which contains a known vulnerability. If exploited, it could lead to denial of service (DoS) when given a large file. More details about the issue can be found at <https://www.npmjs.com/advisories/1556>. The issue was fixed in `node-fetch` v2.6.1 although this newer version is not used by `vue-styled-components`.

Recommendation Remove the dependent `vue-styled-components`. Also, we highly recommend executing `npm audit fix` to check the existence of any 0day vulnerabilities after changing the dependent libraries.

Status This issue has been confirmed by the team.



4 | Conclusion

In this audit, we have analyzed the design and implementation of OneWallet. The audited code does involve various intricacies in both design and implementation. During the audit, we notice that the current code base is well structured and neatly organized, and the identified issues are promptly confirmed and fixed.

In the meantime, we emphasize that even if a wallet is well-designed and securely implemented, it does not guarantee you will not be vulnerable to attacks such as social engineering. Therefore, we strongly encourage wallet users to use common sense and apply basic security methods to keep their digital assets safe.



References

- [1] Harmony. Harmony Inc. <https://harmony.one>.
- [2] MITRE. CWE-1104: Use of Unmaintained Third Party Components. <https://cwe.mitre.org/data/definitions/1104.html>.
- [3] MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- [4] MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.
- [5] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
- [6] PeckShield. PeckShield Inc. <https://www.peckshield.com>.