WEIROLL SECURITY AUDIT REPORT

October 4, 2022

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Customer. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the cients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reoprts into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Customer with an interim report.

5. Bug fixing & re-audit:

- The Customer either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Customer with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Weiroll is a simple and efficient operation-chaining/scripting language for the EVM.

This simple architecture makes it possible for the output of one operation to be used as an input to any other, as well as allowing static values to be supplied by specifying them as part of the initial state.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Enso, Element, Weiroll
Project name	Weiroll
Timeline	5 Sep 2022 - 28 Sep 2022
Number of Auditors	3

Project Log

Date	Commit Hash	Note
05.09.2022	9289118113f753d460a97d7c6ee72577e5c3f5eb	Initial commit
28.09.2022	7770653e40bec8206337bd4e08b5e3e7ef72c0c3	Commit with fixes

Project Scope

The audit covered the following files:

File name	Link
CommandBuilder.sol	CommandBuilder.sol
VM.sol	VM.sol
Events.sol	Events.sol
Math.sol	Math.sol
Strings.sol	Strings.sol
Tupler.sol	Tupler.sol

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	0
Low	4

ID	Name	Severity	Status
L-1	Using `camelCase` style for variables	Low	Fixed
L-2	Unused variables	Low	Fixed
L-3	Merge unchecked blocks	Low	Fixed
L-4	Compiler version (gas saving)	Low	Fixed

1.6 Conclusion

The audited smart contract is an auxiliary code intended to be a part of a more complex project. Thus, some sensitive functionality is not implemented in the audited code base, i.e. access control. During the audit process, we have not found any significant security issues in the supplied code. However, an additional audit of the upcoming code is recommended.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Using `camelCase` style for variables
Files	VM.sol#L43 CommandBuilder.sol#L35
Severity	Low
Status	Fixed in e8d4bb69

Description

Variable naming can be updated:

- VM.sol#L43 `outdata` by `outData`
- VM.sol#L99 `calleth` by `callEth`
- CommandBuilder.sol#L35 `arglen` by `argLen`
- CommandBuilder.sol#L92 `statevar` by `stateVar`
- CommandBuilder.sol#L119 `argptr` by `argPtr`

- CommandBuilder.sol#L169 `srcidx` by `srcIdx`
- CommandBuilder.sol#L171 `destidx` by `destIdx`

Recommendation

We recommend updating the variable naming in favour of code style improvement.

L-2	Unused variables
File	VM.sol#L22
Severity	Low
Status	Fixed in 664e3c23

Description

VM.sol#L22

Recommendation

We recommend removing it or describing its functionality.

L-3	Merge unchecked blocks
File	CommandBuilder.sol#L49
Severity	Low
Status	Fixed in 69129094

Description

Some `unchecked` blocks can be merged:

- CommandBuilder.sol#L49
- CommandBuilder.sol#L97

Recommendation

We recommend merging it in favour of the code readability.

L-4	Compiler version (gas saving)
File	
Severity	Low
Status	Fixed in db07976c

Description

In solidity code and config specified compiler version `0.8.11`, more modern compiler version can improve gas optimization.

Recommendation

We recommend updating it to `0.8.16` for gas saving.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and
software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes