Covalent

AUDIT REPORT

Version 1.0.0

Serial No. 20220329000220224

Presented by Fairyproof

March 29th, 2022

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Covalent project.

Audit Start Time:

March 11, 2022

Audit End Time:

March 24, 2022

Audited Source Files:

There is only one file audited:

```
./
└── OperationalStaking.sol

1 directory, 1 file
```

The goal of this audit is to review Covalent's solidity implementation for its staking function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Covalent team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from off-chain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

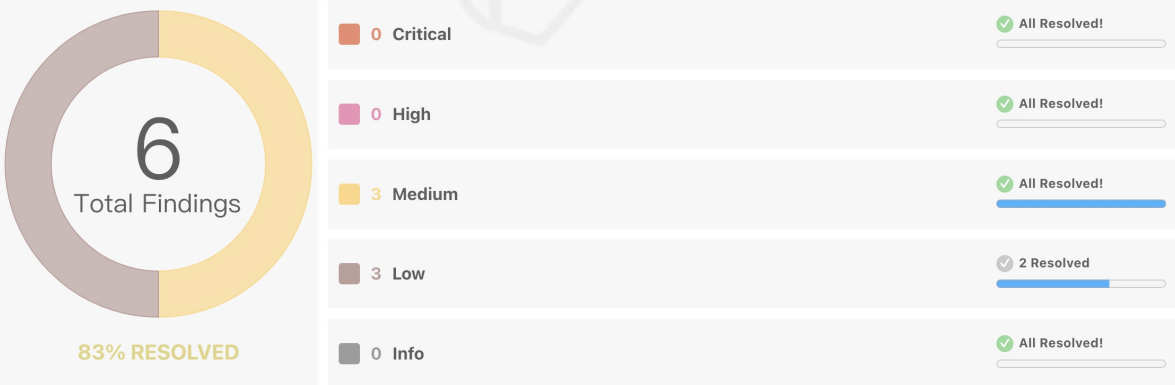
For this audit, we used the following sources of truth about how the staking function should work:

Source File

This was considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Covalent team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2022032900022024	Fairyproof Security Team	2022.03.11 - 2022.03.24	Low



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, 3 risks of medium-severity and 3 risks of low-severity were discovered. The 3 risks of medium-severity and 2 risks of low-severity have been fixed and 1 risk of low-severity has been confirmed.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Major functions of audited code

The audited code mainly implements a staking function. Here are the details:

- Initialization of parameters for staking pools
- Deposit and withdrawal of tokens
- User can stake, unstake tokens, and recover unstaking and redeem rewards
- Add, enable, disable and reward a validator
- Setting of parameters for the application's maintenance and operations.

Note:

- Other functions listed in the white paper were not covered by this audit.

.

04. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-Entrancy Attack
- Replay Attack
- Reordering Attack
- Miner's Advantage
- Rollback Attack
- DDos Attack
- Transaction Ordering Attack
- Race Condition
- Access Control
- Integer Overflow/Underflow
- Timestamp Attack
- Gas Consumption
- Inappropriate Callback Function
- Function Visibility
- Implementation Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Parameter Setting
- Tx.origin
- Fake Deposit
- Shadow Variable
- Design Vulnerability
- Token Issurance

- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration
- Code Improvement
- Misc

05. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

06. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.

We found one issue, please refer to 【FP-1】 in "08. Issue descriptions" for more details.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by "owner".

We didn't find issues or risks in these functions or areas at the time of writing.

- Parameter Setting

We checked whether or not the parameters in this application were set properly.

We found two issues, please refer to 【FP-2】 and 【FP-3】in "08. Issue descriptions".

- State Update

We checked some key state variables which should only be set at initialization.

We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.

We didn't find issues or risks in these functions or areas at the time of writing.

- Contract Migration/Upgrade

We checked whether or not the contract files introduced issues or risks associated with contract migration/upgrade.

We didn't find issues or risks in these functions or areas at the time of writing.

- Implementation of Unstaking

We checked whether or not the unstaking implementation had issues or risks

We found one issue, please refer to 【FP-4】 in "08. Issue descriptions".

- Implementation of Withdrawal

We checked whether or not the implementation of withdrawal had issues or risks.

We found one issue, please refer to 【FP-5】 in "08. Issue descriptions".

- Re-Entrancy Risk

We checked whether or not there were re-entrancy risks when users withdrew assets.

We found one issue, please refer to 【FP-6】in "08. Issue descriptions".

- Miscellaneous

The Fairyproof team didn't find issues or risks in other functions or areas at the time of writing.

07. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Integer Overflow	Integer Overflow/Underflow	Low	✓Fixed
FP-2	Missing Constraints For Parameters	Parameter Setting	Low	✓Fixed
FP-3	Missing Constraints For Parameters	Parameter Setting	Low	Confirmed
FP-4	Improper Implementation	Implementation Vulnerability	Medium	✓Fixed
FP-5	Incorrect Algorithm	Implementation Vulnerability	Medium	✓Fixed
FP-6	Re-Entrancy Risk	Re-Entrancy Attack	Medium	✓Fixed

08. Issue descriptions

[FP-1] Integer Overflow Low ✓Fixed

Issue/Risk: Integer Overflow/Underflow

Description:

In `OperationalStaking.sol`, there were unnecessary `unchecked` statements in lines 92, 100, 128, 269, 355 and 356, which would introduce overflow/underflow issues.

Recommendation:

Consider removing all the 'unchecked' statements.

Status:

It has been fixed by the Covalent team.

[FP-2] Missing Constraints For Parameters

Low

✓Fixed

Issue/Risk: Parameter Setting

Description:

In `OperationalStaking.sol`, the `addValidator` function's `commissionRate` should be less than `DIVIDER` otherwise line 148 would introduce an overflow issue.

Recommendation:

Consider adding the following code in line 121 to validate the `commissionRate` parameter

```
require(commissionRate < DIVIDER, "Rate must be less than 100%");
```

Status:

It has been fixed by the Covalent team.

[FP-3] Missing Constraints For Parameters

Low

Confirmed

Issue/Risk: Parameter Setting

Description:

If the value of `maxCapMultiplier` was reduced too much by the admin, an active validator wouldn't unstake because of line 256 as follows in `OperationalStaking.sol`.

```
require( v.delegated <= newValidatorMaxCap, "Cannot unstake beyond max cap");
```

Recommendation:

Consider adding a check to ensure the value of `maxCapMultiplier` wouldn't affect a validator's unstaking.

Status:

It has been confirmed by the Covalent team. The team replied that it would check the value carefully before assigning it to the parameter.

[FP-4] Improper Implementation

Medium

✓Fixed

Issue/Risk: Implementation Vulnerability

Description:

The value of `delegated` of an inactive validator could be reduced in the following code in `v.delegated -= amount`, but in line 231, the value of `v.delegated` of a non-validator's staking could be increased. That would be contradicted. In addition, the formula used to get `newValidatorMaxCap` was incorrect.

```
if (isValidator && v.disabledAtBlock == 0) {
  // validators will have to disable themselves if they want to unstake tokens
below delegation max cap
  uint128 newValidatorMaxCap = s.staked * maxCapMultiplier;
  require( v.delegated <= newValidatorMaxCap, "Cannot unstake beyond max cap");
}
else {
  v.delegated -= amount;
}
```

Recommendation:

Consider changing the 'if else' logic flow and using the correct formula to calculate `newValidatorMaxCap`.

```
if (isValidator && v.disabledAtBlock == 0) {
  // validators will have to disable themselves if they want to unstake tokens
below delegation max cap
  uint128 newValidatorMaxCap = (s.staked - amount) * maxCapMultiplier;
  require( v.delegated <= newValidatorMaxCap, "Cannot unstake beyond max cap");
}

if(!isValidator) {
  v.delegated -= amount;
}
```

Status:

It has been fixed by the Covalent team.

[FP-5] Incorrect Algorithm

Medium ✓Fixed

Issue/Risk: Implementation Vulnerability

Description:

An inactive validator's withdrawal cooldown time should be counted as starting from the value of `disabledAtBlock` instead of `block.number - disabledAtBlock`

```
  uint128 coolDownEnd = uint128( v.disabledAtBlock != 0 ? block.number -
v.disabledAtBlock : block.number);
```

Recommendation:

Consider changing `block.number - v.disabledAtBlock` to `v.disabledAtBlock`.

```
uint128 coolDownEnd = uint128(v.disabledAtBlock != 0 ? v.disabledAtBlock :
block.number);
```

Status:

It has been fixed by the Covalent team.

[FP-6] Re-Entrancy Risk

Medium ✓Fixed

Issue/Risk: Re-Entrancy Attack

Description:

A token transfer should happen after data modification was completed. This is to avoid re-entrancy attacks.

```
_transferFromContract(msg.sender, amount);
```

Recommendation:

Consider moving line 293 to line 296 to prevent re-entrancy attacks

Status:

It has been fixed by the Covalent team.

09. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- N/A

Appendix

Unit Test Result:

97.18% Statements 172/177

85.23% Branches 75/88

100% Functions 30/30

97.7% Lines 170/174

File		Statements		Branches		Functions		Lines	
contracts/		97.14%	170/175	85.23%	75/88	100%	29/29	97.67%	168/172
contracts/mock/		100%	2/2	100%	0/0	100%	1/1	100%	2/2

Audited File's SHA-256 Value:

```
OperationalStaking.sol:0x3e5e82f0da8ce5d189e329e76088e57d2002ed506087590d515415a
1b4c688f7
```

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech