safepress

Smart Contract Audit

For smart contract vulnerabilities,
security exploits and attack vectors

safepress

SAFEPRESS LTD
3 FRASER STREET #05-25 DUO TOWER SINGAPORE

Code Review And Security Report

Important: This document likely contains critical information about the Client's software and hardware systems, security susceptibilities, descriptions of possible exploits and attack vectors. The document shall remain undisclosed until any significant vulnerabilities are remedied.

CLIENT:	CovalentHQ
START DATE:	23rd August 2022
END DATE:	13th September 2022
TYPE, SUBTYPE:	Staking

Scope

Repository:	https://github.com/covalenthq/bsp-staking
Commit hashes:	cbbe03cabc4ef0a2d32516c252f30c72ed991a20 (incl.)
Documentation:	bsp-staking readme, Covalent Whitepaper May 2022 v1.1, Branded, Block Specimen Whitepaper V1.2
Tests:	Passing
Auditors:	Alex, Rony
Review & Approval:	Ruby
Smart Contract Audited:	contracts/OperationalStaking.sol

Definitions of vulnerability classification

Severity	Definition
Critical	Bug / Logic failures in the code that cause loss of assets / data manipulation.
High	Difficult to exploit problems which could result in elevated privileges, data loss etc.
Medium	Bug / Logic failures in the code which need to be fixed but cannot lead to loss of assets / data manipulation.
Low	Mostly related to unused code, style guide violations, code snippets with low effect etc.

The smart contracts were found with the following vulnerabilities:

Critical Vulnerabilities

- [ ] None

High-level Vulnerabilities

- **Highly permissive owner access.**

  Function (s): setMaxCapMultiplier, setValidatorMaxStake, setValidatorCommissionRate

  Description: The owner can reset *maxCapMultiplier*, *validator MaxStake* and *validator commission rate*. The owner should not have the ability to reset these values after they have been initialized.

  Recommendation: remove the functions that give access to the owner to reset the values after they have already been initialized.

  Status: Design Choice

  Update Note:

  Validator commision rate's initial value is set by the Staking Manager. To change the *commission rate*, the validator should contact the Staking Manager and not the owner.

  Similarly, the owner can change the *multiplier* and the *max stake.*

  This centralizes a lot of authority with the owner. It is recommended to introduce a system where these values are set by the community through voting as planned for Proof Chain (*Covalent Whitepaper May 2022 v1.1 Branded.pdf, page 7, Governance*) or through some other method.

  Update Note: The issue has been acknowledged by the Covalent team and has been ignored as it is a design choice.

- **Possibility of initializing invalid values.**

  Function (s): initialize

  Description: There is a chance of initializing *maxCapMultiplier* and *validatorMaxStake* to wrong values in the *initialize* function

  Recommendation: add require statements in the initialize function to ensure that *maxCapM* and *vMaxStake* are always greater than 0.

  Status: Design Choice

  Update Note: The require statement should still be added as the change is minimal and the issue would persist when the contract is deployed again or to other chains.

  Update Note: The issue has been acknowledged by the Covalent team and has been ignored as it is a design choice.

Medium-level Vulnerabilities

- **Possible invalid validatorId.**

  Function (s): getValidatorStakingData, getValidatorCompoundedStakingData, getDelegatorMetadata, setValidatorAddress.

  Description: It is possible to pass invalid *validatorId.*

  Recommendation: Add require statements to ensure that *validatorId* is always valid. Refer to line 464 where this require statement is already being used.

  Status: Partially Done

  Update Note: The require statement needed to check if an invalid id has been accidentally sent should still be added as the gas consumed for this simple check is minimal.

  Update Note: The require statement has not been added for getValidatorCompoundedStakingData

- **Possible invalid argument *address*.**

  Function (s): setValidatorAddress, setStakingManagerAddress, getDelegatorMetadata

  Description: It is possible to pass invalid argument of type *address* to functions *setValidatorAddress, setStakingManagerAddress* and *getDelegatorMetadata*.

  Recommendation: Add a require statement to ensure that it is always valid. Refer to line 465 where such require statement is already being used.

  Status: Partially Done

  Update Note: The require statements have not been added for *setStakingManagerAddress* and *getDelegatorMetadata*

  Update Note: The require statement has not been added for *getDelegatorMetadata*

Low-level Vulnerabilities

- **Unused imports and variables in tests.**

  Description: There is a large number of unused imports and variables in test files.

  Recommendation: remove unused imports and variables in test files

  Status: Incomplete

- **Improper function documentation.**

  Description: function documentation is very brief and lacks explanation for function parameters.

  Recommendation: make function documentation more descriptive and add variable descriptions e.g

  /*

  * function description comes here

  * @param _param1: param1 explanation that explains what it is for

  * @param _param2: param2 explanation that explains what it is for

  */

  function test(uint _param1, uint _param2) external {

  require(_param1 == 1 && _param2 == 2); }

  Status: Incomplete

Informational

- [ ] None

Executive Summary

Based on the audit findings the Client's contracts are: Well Secured

Not Secure	Insufficiently Secured	Secured	Well Secured

Disclaimers

SafePress Disclaimer
The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer
Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.