safepress

Smart Contract Audit

Covalent HQ BSP Staking Smart Contract

CUSTOMER: Covalent HQ

DATE: Dec 6th, 2022

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:	COVALENT HQ	START DATE:	2022-11-28
TYPE, SUBTYPE:	STAKING	END DATE:	2022-12-06

Scope

REPOSITORY:	https://github.com/covalenthq/bsp-staking
DOCUMENTATION:	https://github.com/covalenthq/bsp-staking#readme
TESTS:	Passing
AUDITORS:	Alex, Rony
REVIEW $ APPROVAL:	Ruby
SMART CONTRACT AUDITED;	contracts / MigrationOperationalStaking.sol

Commit hashes:

BASE:	A6F5CEBCBB313DE60DD24268DFAA62BA993F9C49
Update 1:	48F153C07AA7B6E3C76DBA51220BEE5EDB723435 (INCL.)

Definitions of vulnerability classification

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.

Findings

0	0	0	5	0
Critical	High	Medium	Low	Informational

Summary

Cov-01	Create modifiers for common require statements	Low	Fixed
Cov-02	Missing validation in getValidatorCompoundedStakingData()	Low	Acknowledged
Cov-03	No zero address check	Low	Fixed
Cov-04	Extract the length of the array defaultToBurn in a separate variable	Low	Fixed
Cov-05	Gas inefficiency	Low	Acknowledged

Finding: Cov-01

Create modifiers for common require statements

Base	Function	`_burnDelegatorBalance`	Line 127	Low	Fixed
Base	Function	`getValidatorStakingData`	Line 196	Low	Fixed
Base	Function	`getDelegatorMetadata`	Line 218	Low	Fixed

Description

The contract contains the same require statement, i.e.

```
require(validatorId < validatorsN, "Invalid validator");
```

used in `getValidatorStakingData` `getDelegatorMetadata` and `_burnDelegatorBalance` functions. This means that the same code is repeated in multiple functions.

Recommendation

Create a modifier for the repetitive require statements.

Finding: Cov-02

Missing validation in `getValidatorCompoundedStakingData()`

Base	Function	Line	Severity	Status
	`getValidatorCompoundedStakingData()`	Line 205 ~ 212	Low	Acknowledged

Description

There is no validation for checking that the `validatorId` parameter in `getValidatorCompoundedStakingData()` is valid. This means that the function can be called with an invalid `validatorId` value.

Recommendation

Add a require statement to ensure that `validatorId` is always valid.

Finding: Cov-03

No zero address check

Base	Function	`setCQTAddress`	Line 168	Low	Fixed
Base	Function	`withdrawAllMadCQT`	Line 160	Low	Fixed

Description

The `setCQTAddress` and `withdrawAllMadCQT` functions in the contract does not include any checks to verify that the address parameter is not a zero address (0x0). This could potentially be a risk, as sending funds to the zero address would result in the loss of those funds.

Recommendation

Add require statements to ensure that the address passed to `setCQTAddress` and `withdrawAllMadCQT` functions are valid.

Finding: Cov-04

Extract the length of the array `defaultToBurn` in a separate variable

Base	Function	`burnDefaultDelegators`	Line 110	Low	Fixed

Description

Extract the length of the array `defaultToBurn` used in the condition for the first for loop into a separate variable so that it is not calculated on every loop.

Recommendation

Create a separate variable for the length of the array used in the for loop.

Finding: Cov-05

Gas inefficiency

Base	Function	`burnDefaultDelegators`	Line 101	Low	Acknowledged

Description

The function `burnDefaultDelegators` has nested loops in its code. These loops can cause inefficiency in the amount of gas needed to execute the function, leading to transaction failure. The inefficiency occurs because each iteration of the loop requires additional gas to be consumed. This can cause the total gas required to execute the function to exceed the gas limit set by the Ethereum network, resulting in the transaction being rejected. It is important to carefully consider the use of loops in Solidity code to avoid potential gas inefficiency and transaction failure.

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.