Alcor Staking Liquid Wax

Prepared on: 2024-04-16

Contract: AUD523

Prepared by:
charles Holtzkampf

Prepared for:
Alcor

Table of Contents

Executive Summary

This document outlines any issues found during the audit of the contracts:

- alcorstaking
- bc214d1924aba20a94875e813b210c56247ec697

- The contract has 0 remarks.
- 0 Minor security issues were found.
- 0 Major security issues were found.
- 0 Critical security issues were found.
- The risk associated with this contract is low

REMARK	MINOR	MAJOR	CRITICAL
0	0	0	0

Severity Description

REMARK

Remarks are instances in the code that are worthy of attention, but in no way represent a security flaw in the code. These issues might cause problems with the user experience, confusion with new developers working on the project, or other inconveniences.
Things that would fall under remarks would include:

- Instances where best practices are not followed
- Spelling and grammar mistakes
- Inconsistencies in the code styling and structure

MINOR

Issues of Minor severity can cause problems in the code, but would not cause the code to crash unexpectedly or for funds to be lost. It might cause results that would be unexpected by users, or minor disruptions in operations. Minor problems are prone to become major problems if not addressed appropriately.
Things that would fall under minor would include:

- Logic flaws (excluding those that cause crashes or loss of funds)
- Code duplication
- Ambiguous code

MAJOR

Issues of major security can cause the code to crash unexpectedly, or lead to deadlock situations.
Things that would fall under major would include:

- Logic flaws that cause crashes
- Timeout exceptions
- Incorrect ABI file generation
- Unrestricted resource usage (for example, users can lock all RAM on contract)

CRITICAL

Critical issues cause a loss of funds or severely impact contract usage.
Things that would fall under critical would include:

- Missing checks for authorization
- Logic flaws that cause loss of funds
- Logic flaws that impact economics of system
- All known exploits (for example, on_notification fake transfer exploit)

Methodology

Throughout the review process, we check that the token contract:

- Documentation and code comments match logic and behaviour
- Is not affected by any known vulnerabilities

Our team follows best practices and industry-standard techniques to verify the proper implementation of the smart contract. Our smart contract developers reviewed the contract line by line, documenting any issues as they were discovered. Ontop of the line by line review, we also perform code fuzzzing.

Our strategies consist largely of manual collaboration between multiple team members at each stage of the review, including:

I. Due diligence in assessing the overall code quality of the codebase.
II. Testing contract logic against common and uncommon attack vectors.
III. Thorough, manual review of the codebase, line-by-line.

Our testing includes:

- Overflow Audit
- Authority Control
- Authority Vulnerability
- Re-entry
- Timeout
- RAM Attacks
- Fake contract
- Fake deposit
- Denial of Service
- Design Logic Audit
- RNG attacks
- Stack Injection attacks

Our Code Fuzzing methodology:

We use a modified **EOSIO Contract Development Kit (CDT)** and custom harnessing to make EOS contracts suitable for fuzzing. The contract is instantiated and it's public functions are called in random order, with random input, so as to explore the state space and find corner case inputs that might lead to undesired outcomes. Apart from detecting logic bugs, this approach allows us also to detect memory bugs, hangs, undefined behavior and crash bugs in a semi-automated manner. Since EOS contracts are usually designed to run and complete within a short amount of time, fuzzing them is very fast as well, and therefore an effective instrument for teasing out bugs within the duration of an audit.

MEDIUM

1 Too many withdrawals can cause the contract to timeout

In `alcorstaking.cpp:botclaim()`

During periods of high usage, or if a malicious attacker creates a lot of withdrawals, the function can timeout freezing all withdrawls. This can be mitigated by adding a max iterator count and batching it if necessary

Status: RESOLVED

2 Withdrawals can be returned out as soon as funds have been unstaked as the line to check the minimum duration is commented out

In `alcorstaking.cpp:botclaim()`

The line may need to be updated to a check which continues if the minimum withdrawal time has not passed, else all withdrawals will be locked. The alternate approach is to remove it, but this will throw the exception from the system contract as funds may not be unstaked yet.

Status: RESOLVED