Contract Audit Results

Prepared on: Mar 9, 2022

Contract: AUD463

Prepared by:

Charles Holtzkampf

Sentnlio Ltd

Prepared for:

Bastion

Table of Contents

Executive Summary

This audit was conducted by Sentnl for Bastion. The audit focused on the following contract(s):

- LockdropVault

No issues were found with the contract LockdropVault. The overall assessment is that the risk associated with the LockdropVaultV2.sol contract is low.

Sentnl's audit identified no major or critical security issues.

Project Summary

Name	Bastion
Description	https://www.bastionprotocol.com
Codebase	https://github.com/Near-Lending-Protocol/testnet-deploy/blob/lockdrop-ctokendeposit/contracts/LockdropVaultV2.sol
Platform	ETH
Method of Audit	Static Analysis, Manual Review, Fuzzing
Delivery date	Mar 9, 2022

Contracts in Scope

Contract	Location
LockdropVault	https://github.com/Near-Lending-Protocol/testnet-deploy/blob/lockdrop-ctokendeposit/contracts/LockdropVaultV2.sol

Vulnerability Summary

Total Issues	**0**
Total Critical	0
Total Major	0
Total Minor	0

Total Issues	**0**
Toral Remarks	0

Audit Summary

Audit Type	Findings
Overflow	no issues
Authority Control	no issues
Authority Vulnerability	no issues
Re-entry	no issues
Timeout	no issues
RAM Attacks	no issues
Fake contract	no issues
Fake deposit	no issues
Denial of Service	no issues
Design Logic	no issues
RNG Attacks	no issues
Stack Injection attacks	no issues
General question	no issues

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.

Test Description

Audit Type	Findings
Overflow	Examines where code, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.
Authority Control	Examines the account permissions to assess whether the controlling account has permissions to override the contract.
Authority Vulnerability	Examines account permissions to ensure primary and active keys differ and multisig permissions are not vulnerable.
Re-entry	Action execution can be interrupted in the middle, initiated over (re-entered), and both runs can complete without any errors in execution.
Timeout	Depending on node and or chain settings it's possible that a transaction takes too long to execute, causing it to timeout.
RAM Attacks	Examines whether it's possible for a user to manipulate the contract to use up resources.
Fake contract	Attacker can deploy fake eosio.token contract, and if symbol is checked but contract isn't, then user could deposit fake tokens.
Fake deposit	Examines whether user can send a fake require_recipient triggering a deposit.
Denial of Service	Ram / timeout attacks, forcing the contract off line by spamming it in some way that causes future users to either timeout, or fail as the contract doesn't have enough RAM.
Design Logic	Basic error in the logic of the smart contract that could effect the execution.

Audit Type	Findings
RNG Attacks	If a random number is generated, ensure the entropy isn't deterministic
Stack Injection attacks	Examines whether user can load a fake contrat and inject into the call stack

Audit Results - LockdropVault

LockdropVaultV2.sol	ID	FILE-01
LockdropVaultV2.sol	LOCATION	https://github.com/Near-Lending-Protocol/testnet-deploy/blob/lockdrop-ctokendeposit/contracts/LockdropVaultV2.sol
LockdropVaultV2.sol	SHA256	f0a58ccca333e40beb52dd671eec18230c2f11d6c9a75005e1c14a738a25fd7e