SECURITY AUDIT OF

LUASWAPIDO SMART CONTRACTS

Public Report

Oct 14, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Oct 14, 2021. We would like to thank the Tomochain for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the LuaSwapIDO Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About LuaSwapIDO Smart Contracts

Powered on Ethereum and TomoChain networks, LuaStarter empowers project owners to raise capital on a decentralized platform and provides the most cost-effective, convenient and transparent investing environment to all investors.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the LuaSwapIDO Smart Contracts. It was conducted on the source code provided by the Tomochain team.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The initial review was conducted on Oct 1, 2021 and a total effort of 7 working days was dedicated to identifying and documenting security issues in the code base of the LuaSwapIDO Smart Contracts.

[redacted]

2.2. Findings

The LuaSwapIDO Smart Contracts was written in Solidity language, with the required version to be 0.6.12. The source code was written based on OpenZeppelin's library.

Tomochain fixed the code according to Verichain's draft report.

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

2.2.1. Checking equality of ratios by integer division MEDIUM

Cap modifications ensure a constant ratio of IDO amount / buy amount over modifications by comparing the old ratio with the modification.

[redacted]

This check can be wrong as it is performed using integer division, while the ratio is not defined as integer anywhere. When the ratio changed, errors can be accumulated until large enough they can change the ratio itself. On another hand, the error ratio can make the equivalent IDO token of result buying amount to be greater or smaller than actual amount of token in the IDO contract itself by an amount proportion to the error ratio, this will result in some tokens to be stucked within the IDO contract or missing from IDO contract so that last claiming users will be failed.

RECOMMENDATION

The most simple solution is ensuring integer ratio:

[redacted]

If we can sure that multiplication will not overflow, we can do so: a/b = c/d is equivalent to ad=bc, bd!=0.

Another simple solution is introducing variable for the ratio in reduced fraction, ratioNumerator and ratioDenominator, small enough so that the ratio can be efficiently checked by multiplication, but this method has a tradeoff of new variable and existing code modification.

In case we need to support generic ratio, and the values can be overflowed, we can use an algorithm similar Euclide to compare the ratios using integer division:

[redacted]

Alternatively, the comparision can be done approximately by multiplying numerators by a factor f, errors will be at O(1/f):

[redacted]

The final claimable ido amount will be errored by a proportion of that error, so we can let the last user claim all the remaining token instead (he will gain/lose a small/very small amount of tokens proportion to 10**decimal/F wei).

UPDATES

- *2021-10-14*: This issue has been acknowledged and fixed by the Tomochain team.

2.2.2. Reentrancy with ERC777 pay token MEDIUM

In the removeCommitment function, the transferHelper will be used to return the pay token to the user. However, if the pay token is ERC777, the user can register a tokensReceived callback and exploit the removeCommitment function.

[redacted]

A simple exploit can be summarized as below:

- Creating 2 commit transactions with the same amount.
- Calling removeCommitment function with the above amount. When the pay tokens are returned for the first time, the implementation in the tokensReceived hook should call the removeCommitment again (the second call to the tokensReceived hook should do nothing).
- When the first removeCommitment is completed, the userCommitedAmount will be overwritten, so it is only decreased by amount once.

In summary, with the above exploit, we can get all the committed amount back but the userCommittedAmount is not decreased to zero.

The main reason of this exploit is the usage of update-after-transfer pattern, commited amount storages are updated after transferring, and updated using the cached original value of these amount, so re-entrant calls wont modify any amount storage at all.

RECOMMENDATION

We can use the ReentrancyGuard module by OpenZeppelin to prevent reentrancy call to this function, or change the implement to update-before-transfer pattern.

UPDATES

- *2021-10-14*: This issue has been acknowledged and fixed by the Tomochain team.

2.2.3. Conflicting precondition/definition of userCommited values array LOW

userCommited[index] array is defined as "accept duplicate" in its addition mutator:

[redacted]

Not only definition, the above if body can be triggered as much as possible by commiting a zero amount. On another hand, it is viewed as non-duplicate collection in other places:

[redacted]

RECOMMENDATION

The collection should be defined as non-duplicating and that definitation should be ensured by disabling zero amount commit or extending the if condition to commitedAmount == 0 && amount > 0.

UPDATES

- *2021-10-14*: This issue has been acknowledged by the Tomochain team.

2.2.4. Reusable offchain signature LOW

The verifyProof function is used to verify the legality parameters sent from backend. However, this function does not check if the proof was used or not, attacker can replay the message many times until expired.

[redacted]

RECOMMENDATION

The proof should be checked to ensure that it is used once.

UPDATES

- *2021-10-14*: This issue has been acknowledged by the Tomochain team.

2.2.5. Incorrect totalAmountPay lower bound checks LOW

The totalAmountPay lower bound checks in the decreaseCap and commit function are incorrect.

[redacted]

Above decreaseCap function can decrease totalAmountPay below totalCommittedAmount. Assume that the first require (amountIDO / amountPay ratio check) is correct, totalAmountIDO will be decreased by the same ratio, to be below the required amount of IDO token for corresponding totalCommittedAmount, that will prevent the last users from claiming their token as the IDO token is not enough at that time.

The correct check should be adding a condition: totalCommittedAmount <= new_totalAmountPay. But in this case if we can guarantee/assume of precondition committedAmount of committing user to be above minAmountPay, we can safely remove the original check, as these 2 will be combined into max(totalCommittedAmount, minAmountPay * #users commited) <= new_totalAmountPay, which is just the new condition because minAmountPay * #users commited <= totalCommittedAmount from precondition.

2.2.5.1. Recommendation.

The checks should be updated as below.

[redacted]

UPDATES

- *2021-10-14*: This issue has been acknowledged by the Tomochain team.

2.3. Additional notes and recommendations

2.3.1. payToken must not be a fee-collecting token INFORMATIVE

Everything will get very bad if payToken is a fee-collecting token, user can drain IDO's balance by an amount equal to his own lost by keeping calling commit and removeCommitment.

RECOMMENDATION

Especially define/document that payToken is not a fee-collecting token, or add logics to handle fee.

UPDATES

- *2021-10-14*: This issue has been acknowledged by the Tomochain team.

2.3.2. Typos INFORMATIVE

There're some typos in the name of these variables.

[redacted]

UPDATES

- *2021-10-14*: This issue has been acknowledged by the Tomochain team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*2021-10-14*	Private Report	Verichains Lab
1.1	*2021-10-14*	Public Report	Verichains Lab

Table 2. Report versions history