SECURITY AUDIT OF

LOOPSTARTER SMART CONTRACTS

Public Report

April 07, 2022

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 April 07, 2022. We would like to thank the LOOPStarter 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 LOOPStarter 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 smart contract code, along with some recommendations. LOOPStarter team has resolved and updated all issues following our recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY **5**

1.1. About LOOPStarter Smart Contracts **5**

1.2. Audit scope **5**

1.3. Audit methodology **5**

1.4. Disclaimer **6**

2. AUDIT RESULT **7**

2.1. Overview **7**

2.1.1. LoopsToken contract 7

2.1.2. DEX contracts 7

2.2. Findings **7**

2.2.1. LOOPSToken.sol - Token contract is vulnerable to sandwich attacks HIGH 8

2.2.2. LOOPSToken.sol - loopsTokenStrategy should be initialized inside the initialize function LOW 9

2.3. Additional notes and recommendations **10**

2.3.1. LOOPSToken.sol - Unused delegate logic INFORMATIVE 10

2.3.2. LOOPSToken.sol - Router address doesn't need to be excluded INFORMATIVE 11

3. VERSION HISTORY **12**

1. MANAGEMENT SUMMARY

1.1. About LOOPStarter Smart Contracts

LOOPStarter is a platform used to launch crypto projects, introduce some new coins, and increase liquidity. This is one of the biggest things for this digital world, especially when it comes to decentralized finance. LOOPStarter was launched on a decentralized exchange - LOOPDEX using LOOPS tokens, which has become an exchange and a crowdfunding platform for all new projects.

For a blockchain-based approach, people from many backgrounds can contribute. LOOPStarter offers regular investors "many opportunities for launchpads that were made available to venture capitalists and some large-scale investors."

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the LOOPStarter Smart Contracts. It was conducted on commit f047027175f8bcc182505128b2425c59bb988a6f from git repository *https://github.com/loopstarter/smartcontracts/*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
f0860fa0c69ecd5226550d1f2b25d9731fd5a71caaa1aa3fadf473ee3139c8e3	**./dex/Factory.sol**
ad033dd0989d30963449c36623bc7c3b68951b314a58d87c0d77a4e82ca1f2c3	**./dex/Router.sol**
d2e6075df877b464cfea91e49f642bcee19580a7c5500b6074e9ef0a830fcb67	**./tokens/LOOPSToken.sol**

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 LOOPStarter Smart Contracts was written in Solidity language based on the OpenZeppelin's library.

There are two main parts in the audit scope as shown in the below sections:

2.1.1. LoopsToken contract

This is the main ERC20 token in the LOOPStarter ecosystem. Besides the default ERC20 functions, the contract implements some additional features like token transfer tax fee (5% on each transfer), automated liquidity acquisition, automated burning, and anti whale. When the token contract is initialized, all the tokens (max supply amount) will be minted and transferred to the contract operator wallet (who is also the contract deployer).

In addition, this token contract is upgradable, so be aware that the admin can change its logic or tax rate ratios at any time.

The below table lists some properties of the audited LoopsToken contract (as of the report writing time).

PROPERTY	VALUE
Name	Loopstarter
Symbol	LOOPS
Decimals	18
Max Supply	100,000,000 (x10^{18})

Table 2. LOOPS token properties

2.1.2. DEX contracts

These contracts provide a Decentralized Exchange (DEX) platform for token swapping. The source code is based on PancakeSwap DEX with some minor modifications.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of LOOPStarter Smart Contracts. LOOPStarter team has 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. LOOPSToken.sol - Token contract is vulnerable to sandwich attacks HIGH

In the swapTokensForEth and addLiquidity functions, since the slippages are not set properly, so this contract may be vulnerable to sandwich attacks. In these attacks, the attackers try to manipulate the token price before the victim's transaction is completed and then get profit.

```
function swapTokensForEth(uint tokenAmount) private {
    // generate the LOOPSSwap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = loopsSwapRouter.WETH();

    _approve(address(this), address(loopsSwapRouter), tokenAmount);

    // make the swap
    loopsSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this),
        block.timestamp
    );
}

function addLiquidity(uint tokenAmount, uint ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(loopsSwapRouter), tokenAmount);

    // add the liquidity
    loopsSwapRouter.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        operator(),
        block.timestamp
    );
}
```

RECOMMENDATION

A safer way is just allowing contract operators to trigger the swapAndLiquify function. If this function is called manually, contract operators can easily input the minimum token amount based on the current price of the token pair.

UPDATES

- *Apr 07, 2022*: This issue has been acknowledged and fixed by the LOOPStarter team in commit af401fa7e65d43984d794579e1ed029bfec661d5.

2.2.2. LOOPSToken.sol - loopsTokenStrategy should be initialized inside the initialize function LOW

The current token contract both has the constructor and initialize functions, which is not correct. In the upgradable contract, all the initialization logic should be added inside the initialize function and not the constructor.

```
constructor(address _loopsTokenStrategy) {
    loopsTokenStrategy = _loopsTokenStrategy;
}

function initialize() external virtual initializer {
    transferTaxRate = 500;
    MAXIMUM_TRANSFER_TAX_RATE = 1000;
    BURN_ADDRESS = 0x000000000000000000000000000000000000dEaD;
    burnRate = 25;
    LPRate = 25;
    maxTransferAmountRate = 50;
    minAmountToLiquify = 500 ether;
    __ERC20_init("Loopstarter","LOOPS");
    _mint(_msgSender(), MAX_SUPPLY);
    __Ownable_init();
    _operator = _msgSender();
    emit OperatorTransferred(address(0), _operator);

    _excludedFromAntiWhale[msg.sender] = true;
    _excludedFromAntiWhale[address(0)] = true;
    _excludedFromAntiWhale[address(this)] = true;
    _excludedFromAntiWhale[BURN_ADDRESS] = true;

    _excludedFromTax[msg.sender] = true;
    _excludedFromTax[loopsTokenStrategy] = true;
    _excludedFromTax[address(this)] = true;
}
```

RECOMMENDATION

Move the loopsTokenStrategy initialization to the initialize function.

UPDATES

- *Apr 07, 2022*: This issue has been acknowledged and fixed by the LOOPStarter team in commit af401fa7e65d43984d794579e1ed029bfec661d5.

2.3. Additional notes and recommendations

2.3.1. LOOPSToken.sol - Unused delegate logic INFORMATIVE

In the LOOPSToken contract, there are some unused code snippets related to the delegation logic as below:

```
// ...
mapping (address => address) internal _delegates;

/// @notice A checkpoint for marking number of votes from a given block
struct Checkpoint {
    uint32 fromBlock;
    uint votes;
}

/// @notice A record of votes checkpoints for each account, by index
mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

/// @notice The number of checkpoints for each account
mapping (address => uint32) public numCheckpoints;

// ...
function delegates(address delegator)
external
view
returns (address)
{
    return _delegates[delegator];
}

/**
 * @notice Delegate votes from `msg.sender` to `delegatee`
    * @param delegatee The address to delegate votes to
    */
function delegate(address delegatee) external {
    return _delegate(msg.sender, delegatee);
}

// ...
```

RECOMMENDATION

These unused code snippets should be removed since it's not related to the current token contract logic.

UPDATES

- *Apr 07, 2022*: This issue has been acknowledged and fixed by the LOOPStarter team in commit af401fa7e65d43984d794579e1ed029bfec661d5.

2.3.2. LOOPSToken.sol - Router address doesn't need to be excluded INFORMATIVE

In the updateloopsSwapRouter function, the _router address is excluded from tax. However, the router contract doesn't hold any tokens when swapping and adding liquidity. All the token transfer operations are just related to the token pair address.

```
function updateloopsSwapRouter(address _router) public onlyOperator {
    loopsSwvapRouter = IRouter2(_router);
    LOOPSSwapPair = ILoopsFactory(loopsSwapRouter.factory()).getPair(addr…
  ess(this), loopsSwapRouter.WETH());
    require(LOOPSSwapPair != address(0), "LOOPS::updateloopsSwapRouter: I…
  nvalid pair address.");
    _excludedFromTax[LOOPSSwapPair] = true;
    _excludedFromTax[_router] = true;
    _excludedFromAntiWhale[_router] = true;
    emit loopsSwapRouterUpdated(msg.sender, address(loopsSwapRouter), LOO…
  PSSwapPair);
}
```

RECOMMENDATION

We should remove the _router address from the _excludedFromTax map.

UPDATES

- *Apr 07, 2022*: This issue has been acknowledged and fixed by the LOOPStarter team in commit af401fa7e65d43984d794579e1ed029bfec661d5.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 07, 2022*	Public Report	Verichains Lab

Table 3. Report versions history