verichains

SECURITY AUDIT OF

SIPHER TOKEN SMART CONTRACT

Public Report

Dec 05, 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 Dec 05, 2021. We would like to thank the Sipher 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 Sipher Token Smart Contract. 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 no vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Sipher Token Smart Contract

Sipher is an ambitious casual fighting and exploration Game with an End-game goal of creating an open world social experience, Built on the Ethereum Blockchain.

10,000 unique NFTs are beautifully illustrated by our 2D & 3D artists. Inspired by the cryptographic ethos, this collection is aptly named "Sipherian Surge", with all the 10,000 characters belonging to the Sipher's origin race: SIPHER INU.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Sipher Token Smart Contract.

The initial review was conducted on Nov 30, 2021 and a total effort of 5 working days was dedicated to identifying and documenting security issues in the code base of the Sipher Token Smart Contract.

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

SHA256 SUM	FILE
b819b4fbc64e21bfafb63ce43504595c01973857ea3612a24a045cbbec75f865	SipherIBCO.sol
243d2d3a4daaff77e60ddcc23201f6ad0609502bc4652dfc621f42facd140b36	SipherToken.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

2.1.1. Sipher Token

This table lists some properties of the audited Sipher Token Smart Contract (as of the report writing time).

PROPERTY	VALUE
Decimals	18
Total Supply	1,000,000,000 ($x10^{18}$) Note: the number of decimals is 18, so the total representation token will be 1,000,000,000 or 1 billion.

Table 2. The Sipher Token Smart Contract properties

2.1.2. Sipher IBCO

Sipher IBCO Smart contract supports the public sale based on Sipher's customized version of the Initial Bonding Curve Offering (IBCO). The goal of IBCOs is to solve the aforementioned problems by utilizing bonding curves of Decentralized Exchanges (DEXs). In the contract, 40.000.000 Sipher Tokens will be for sale. Each time funds are contributed, the settlement price of the token will increase.

There are 2 phases in the public sale:

- Ongoing: Users can deposit ethers and withdraw ethers that they have funded in this phase.
- Ended: In this phase, users can claim tokens. The owner of the contract can withdraw ethers, also he can withdraw the remaining Sipher tokens after 30 days from the end of the sale if users do not claim.

The public sale will start on Monday, December 6, 2021, at 1:00 AM UTC, and end on Thursday, December 9, 2021, at 1:00 AM UTC.

2.2. Contract codes

The Sipher Token Smart Contract was written in Solidity language, with the required version to be ^0.8.7.

The source codes consist of three contracts, one abstract contract, two libraries. Almost all source codes in the Sipher Token Smart Contract imported OpenZeppelin contracts.

2.2.1. ERC20 contract

This is the contract implement ERC20 token. The source code is referenced from OpenZeppelin's implementation.

2.2.2. Ownable abstract contract

Contract module which provides a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions. The source code is referenced from OpenZeppelin's implementation.

2.2.3. SafeERC20 library

This is the wrappers around ERC20 operations that throw on failure (when the token contract returns false). Tokens that return no value (and instead revert or throw on failure) are also supported, non-reverting calls are assumed to be successful. The source code is referenced from OpenZeppelin's implementation.

2.2.4. Math library

This is the standard math utilities missing in the Solidity language. The source code is referenced from OpenZeppelin's implementation.

2.2.5. SipherToken contract

This is the token contract in the Sipher Token Smart Contract, which is extended from ERC20 contract.

2.2.6. SipherIBCO contract

This is the main contract in the Sipher Token Smart Contract, which implement Initial Bonding Curve Offering. Users can deploy ETH into contract and claim SIPHER tokens when the offering end.

2.3. Findings

The Sipher Token Smart Contract was written in Solidity language, with the required version to be ^0.8.7. The source code was written based on OpenZeppelin's library.

2.3.1. SipherIBCO.sol - nonsense calculation in _withdrawCap INFORMATIVE

Consider the provided snippet:

```
/**
 * @dev Calculate withdrawCap based on accumulated ether
 */
```

```
function _withdrawCap(uint256 userAccumulated) internal pure returns (uin…
  t256 withdrawableAmount) {
    if (userAccumulated <= 1 ether) {
        return userAccumulated;
    }
    if (userAccumulated > 1 ether && userAccumulated <= 150 ether) {
        uint256 accumulatedTotalInETH = userAccumulated / (10**18);
        uint256 takeBackPercentage = (3 * accumulatedTotalInETH**2 + 7089…
  7 - 903 * accumulatedTotalInETH) / 1000;
        return (userAccumulated * takeBackPercentage) / 100;
    }
    if (userAccumulated > 150 ether) {
        return (userAccumulated * 3) / 100;
    }
}
```

Mathematically, the above function implements a piecewise function according to userAccumulated x (ETH):

- x * 100%, x<=1
- x * (3X^2 + 70 897 - 903X)/100 % where X=floor(x), 1<x<=150
- x * 3%, 150<x

We do not know from where the quadratic coefficients come from. The resulting graph is not continuous, consists of many increasing small continuous subgraphs. The graph in general increases when x<54, decreases almost 4 times from approximately 1700% to 450%, and then linearly increases again. That's a strange graph that we have no idea to understand.

Reference graph: *https://www.desmos.com/calculator/b6549jv0ig*.

UPDATES

- *Dec 03, 2021*: This issue has been acknowledged by the Sipher team.

2.3.2. SipherIBCO.sol - hardcoded decimal for distribute token INFORMATIVE

The contract is hardcoding decimal for distribute token.

```
uint256 public constant TOTAL_DISTRIBUTE_AMOUNT = 40000000 * 10**18;

function getEstTokenPrice() public view returns (uint256) {
    return (Math.max(totalProvided, MINIMAL_PROVIDE_AMOUNT) * 10**18) / T…
  OTAL_DISTRIBUTE_AMOUNT;
}
```

RECOMMENDATION

Using IERC20Metadata to get token decimals for calculating or clearly states that the contract is using Sipher Token - which has decimal equals 18.

UPDATES

- *Dec 03, 2021*: This issue has been acknowledged and fixed by the Sipher team.

2.3.3. SipherToken.sol - _releasableAmount function simplification INFORMATIVE

In _releasableAmount function, many if-else is used but the conditions are duplicated, for example in the original code:

```
if (vestingPoint <3) {
    uint256 vestingOffset =  55000000*DECIMAL;
    return vestingOffset + vestingPoint*(7727273*DECIMAL)-_released;
} else if (vestingPoint >= 3 && vestingPoint <12) {
    uint256 vestingOffset =  70454546 *DECIMAL;
    return vestingOffset + (vestingPoint-2)*(7977273*DECIMAL)-_released;
} else if (vestingPoint >= 12 && vestingPoint <15) {
...
```

RECOMMENDATION

We can clearly simplify the ifs into

```
if (vestingPoint <3) {
    uint256 vestingOffset =  55000000*DECIMAL;
    return vestingOffset + vestingPoint*(7727273*DECIMAL)-_released;
} else if (vestingPoint <12) {
    uint256 vestingOffset =  70454546 *DECIMAL;
    return vestingOffset + (vestingPoint-2)*(7977273*DECIMAL)-_released;
} else if (vestingPoint <15) {
```

UPDATES

- *Dec 03, 2021*: This issue has been acknowledged and fixed by the Sipher team.

APPENDIX

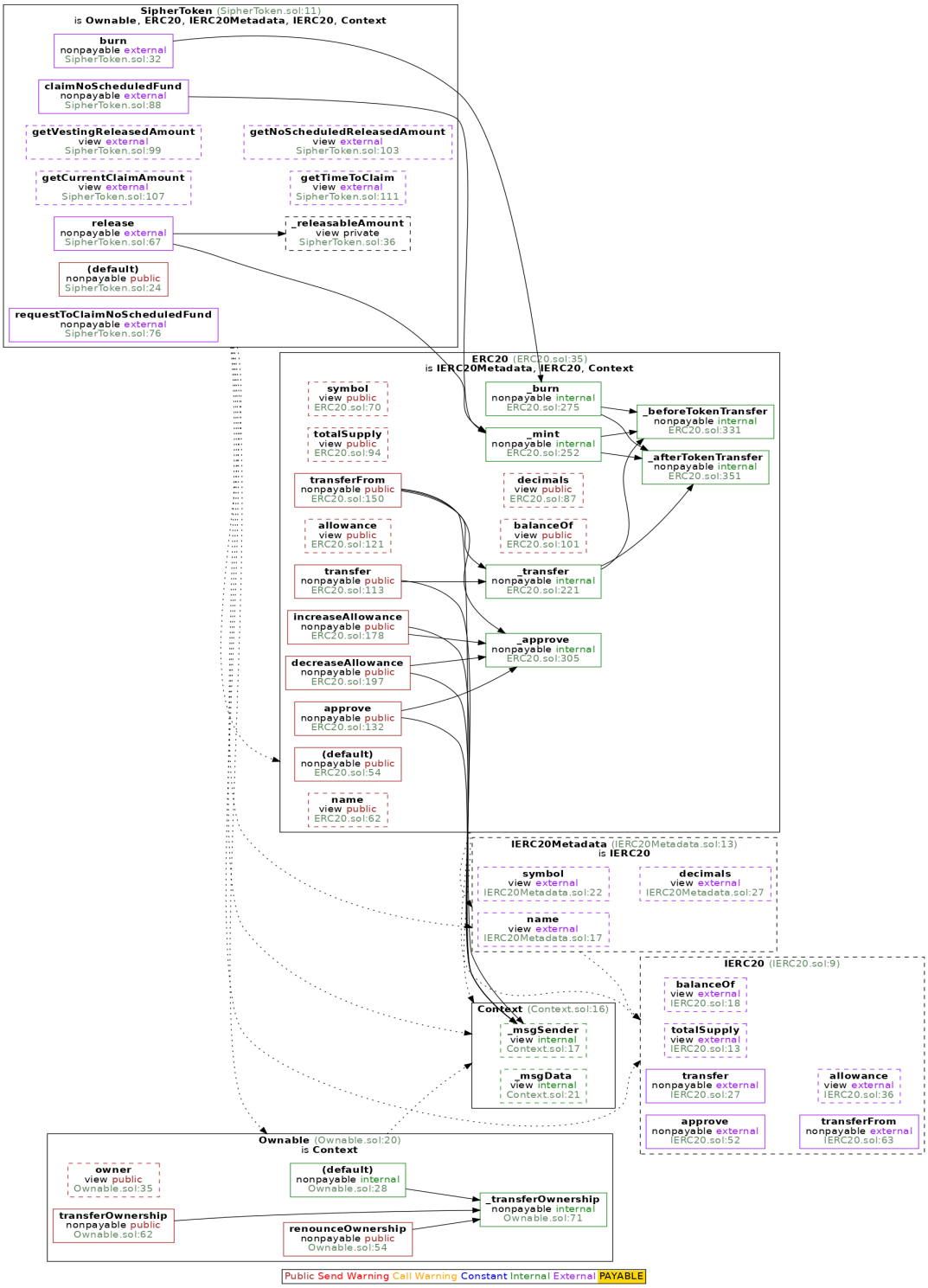

Image 1. Sipher TokenSmart Contract call graph

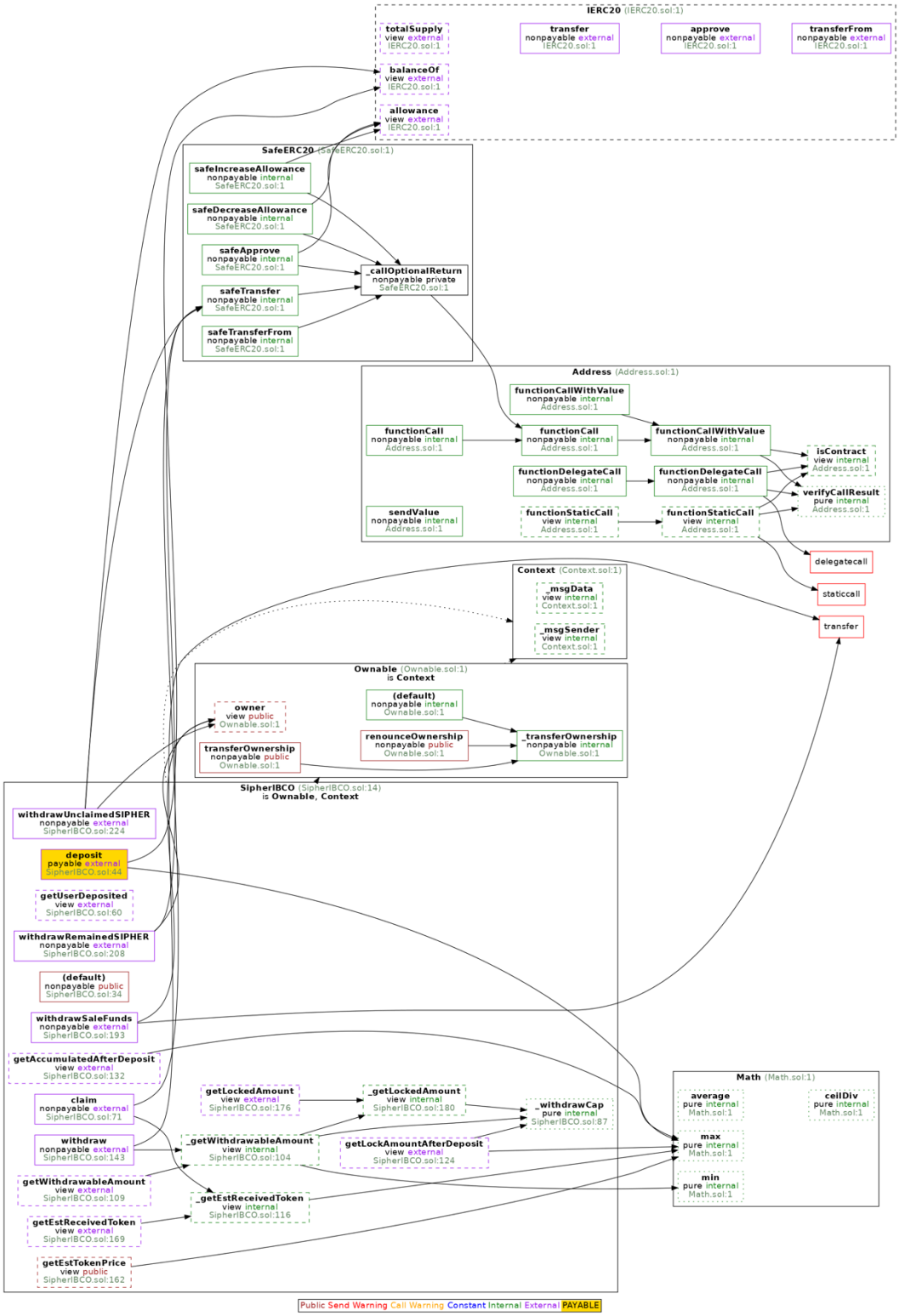

Image 2. Sipher IBCO Smart Contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec 02, 2021*	Public Report	Verichains Lab
1.1	*Dec 03, 2021*	Public Report	Verichains Lab
1.2	*Dec 05, 2021*	Public Report	Verichains Lab

Table 3. Report versions history