verichains

SECURITY AUDIT OF

TAUREUM WITHDRAW SMART CONTRACTS

Public Report

Apr 26, 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 Apr 26, 2022. We would like to thank the Taureum 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 Taureum Withdraw 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 no vulnerable issue in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Taureum Withdraw Smart Contracts

TAUREUM is a decentralized ecosystem based on Taureum Token with the aim of bringing decentralization & blockchain technology to the mass public. In order to achieve its goal, TAUREUM aims to develop a set of products that fully serve the needs of users, from cryptocurrency transactions, value storage, entertainment, education to asset management. TAUREUM was founded with a vision of creating its own blockchain platform to stimulate further the mass public adoption of cryptocurrency & blockchain technology.

1.2. Audit scope

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

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

SHA256 Sum	File
6ded65168255caf27323cd383d7b11fa08a35fd8bb56de483d98c113e5ea1f70	**Withdraw.sol**
fdff068448a72a1b49748efcff80e9586df8af67aecd6325af5a262e4d3e5260	**WithdrawHero.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 Taureum Withdraw Smart Contracts was written in Solidity language, with the required version to be ^0.8.2. The source code was written based on OpenZeppelin's library.

There are two main contracts in the audit scope. They are Withdraw contract and WithdrawHero contract.

2.1.1. Withdraw contract

This contract supports the Taureum team to release the tokens. The claimed users may be set by the owner of contract. Every claiming transaction must be sign by the Taureum team through signature.

2.1.2. WithdrawHero contract

This contract supports the Taureum team to release the NFT tokens. The claimed users may be set by the owner of contract. Every claiming transaction must be sign by the Taureum team through signature.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Taureum Withdraw Smart Contracts

2.3. Additional notes and recommendations

2.3.1. Withdraw.sol, WithdrawHero.sol - Missing event when adding and removing singer INFORMATIVE

In both audit contracts, the signers are the important state variables. But the adding and removing singer functions in the contracts miss the emit events statements to notice successful actions.

We recommend adding the emit event statements to these functions in both contracts.

UPDATES

- *Apr 26, 2022*: This issue has been acknowledged by the Taureum team.

2.3.2. Withdraw.sol, WithdrawHero.sol - Best-practice to avoid reentrancy

INFORMATIVE

```
function claim(
        address to,
        uint256 amount,
        uint256 nonce,
        bytes calldata signature
    ) external {

        bytes32 _msgHash = keccak256(
            abi.encodePacked(
                "\x19Ethereum Signed Message:\n32",
                keccak256(abi.encodePacked("OT_WITHDRAW_TAUM", to, amount…
  ,nonce))
            )
        );
        address signer = getSigner(_msgHash, signature);
        require(signers[signer], "OT: invalid signer");

        require(!isClaimed[signature],"OT: token claimed");
        require(totalClaim[block.timestamp/timeUint] + amount <= maximumP…
  erUintTime, "OT: over quota");
        require(lastTimeClaim[to] + lockTimeWithdrawPerUser < block.times…
  tamp, "OT: limit time withdraw");

    taumToken.safeTransferFrom(storeTaum, to, amount);
        isClaimed[signature] = true;
        totalClaim[block.timestamp/timeUint] += amount;
        lastTimeClaim[to] = block.timestamp;
        emit ClaimToken(to,amount,nonce,signature);
    }
```

The claim function in both audit contracts misses checking the reentrancy state. If the target contract implements the logic to trigger the receiver, the contract may be attacked.

However, with the target contract used and the logic claim function in the contract, it's not a problem. But we still create this issue to notice Taureum avoid the problems in the development.

RECOMMENDATION

We suggest extending the contracts with ReentrancyGuard abstract contract and using nonReentrant modifier for the claim function.

UPDATES

- *Apr 26, 2022*: This issue has been acknowledged by the Taureum team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 26, 2022*	Private Report	Verichains Lab

Table 2. Report versions history