verichains

SECURITY AUDIT OF

TAUREUM SMART CONTRACTS

Public Report

Dec 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 Dec 14, 2021. 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 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 issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Taureum 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 Smart Contracts. It was conducted on commit f6faf5fa4c4a87f27867ea263dfc926e80125798 from git repository *https://github.com/f6galaxy/orbitau-smartcontracts/*.

There are 2 files in our audit scope. They are Tau.sol, OrbiTauVestingFactory.sol files.

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 Dec 10, 2021 and a total effort of 3 working days was dedicated to identifying and documenting security issues in the code base of the Taureum Smart Contracts.

The following files were made available in the course of the review: Tau.sol and OrbiTauVestingFactory.sol.

2.2. Contract code

The Taureum 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.

2.2.1. Tau token contract

Tau contract is an ERC20 token contract. Besides the default ERC20 functions, the contract implements additional functions which avoid bot trading to control the price of tokens in the IDO time.

2.2.2. OrbiTauVestingFactory contract

OrbiTauVestingFactory is a contract that inherits the VestingWallet contract. In the contract, the owner can create new VestingWallets that release the tokens following the tokenomics.

2.3. Findings

During the audit process, the audit team found no vulnerability in the given version of Taureum Smart Contracts.

2.4. Additional notes and recommendations

2.4.1. Tau.sol - BPContract function INFORMATIVE

Since we do not control the logic of the BPContract, there is no guarantee that BPContract will not contain any security related issues. With the current context, in case the BPContract is compromised, there is not yet a way to exploit the Taureum Smart Contracts, but we still note that here as a warning for avoiding any related issue in the future.

By the way, if having any issue, the BPContract function can be easily disabled anytime by the contract owner using the setBpEnabled function. In addition, BPContract is only used in a short time in token public sale IDO then the contract owner will disable it forever by the setBotProtectionDisableForever function.

2.4.2. OrbiTauVestingFactory.sol - Gas optimize in createVestingToken INFORMATIVE

In the createVestingToken function, the saleConfig variable is only used for getting data. Using memory declaration for saleConfig will force Solidity to copy it from storage to memory which will cost more gas than using from storage.

RECOMMENDATION

Consider change memory to storage for this declaration contracts for gas saving.

Example:

```
function createVestingToken(
        address beneficiaryAddress,
        uint256 amount,
        uint256 saleId
    ) external onlyOwner {
        require(
            beneficiaryAddress != address(0),
            "OT: invalid beneficiary address"
        );
        require(saleId < sales.length,"OT: invalid saleId");

        SaleConfig storage saleConfig = sales[saleId];

        require(saleConfig.isActive,"OT: saleId is inactive");

        require(
            vestingAddresses[saleId][beneficiaryAddress] == address(0),
            "OT: exits vesting "
        );
        require(
            IERC20(saleConfig.tokenAddress).allowance(saleConfig.holderTo…
  ken, address(this)) >=
                amount,
            "OT: required approve"
        );
        uint64 startTimestamp = saleConfig.startTime + saleConfig.cliff ;
        uint64 durationSeconds = saleConfig.duration ;

        VestingWallet vestingAddress = new VestingWallet(
            beneficiaryAddress,
```

```
            startTimestamp,
            durationSeconds
        );

        vestingAddresses[saleId][beneficiaryAddress] = address(vestingAdd…
ress);

        SafeERC20.safeTransferFrom(
            IERC20(saleConfig.tokenAddress),
            saleConfig.holderToken,
            address(vestingAddress),
            amount
        );

        emit VestingCreate(
            beneficiaryAddress,
            address(vestingAddress),
            amount,
            saleId
        );
    }
```

Snippet 1. OrbiTauVestingFactory.sol Recommend fixing in `createVestingToken` function

UPDATES

- *Dec, 14,2021*: This issue has been acknowledged and fixed by the Taureum team in commit cedfacd8ec4c570e6e4dd8beed73e847fa079c5d.

APPENDIX

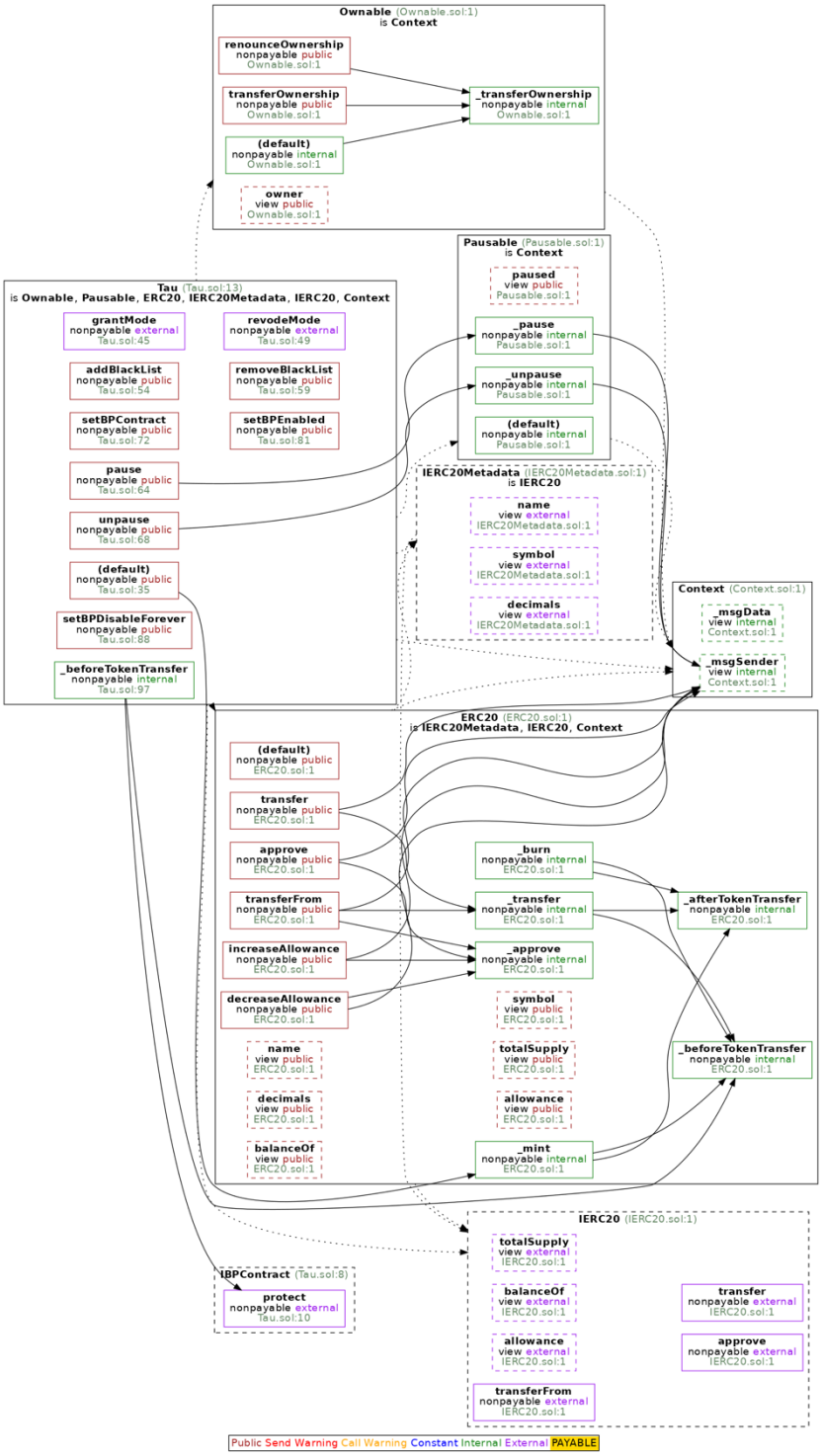

Image 1. Taureum Smart contracts call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Dec, 14,2021*	Public Report	Verichains Lab

Table 2. Report versions history