verichains

SECURITY AUDIT OF

TANKWARSZONE TOKEN AND TOKENVESTING SMART CONTRACTS

Public Report

Dec 02, 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 02, 2021. We would like to thank the Tankwarszone 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 Tankwarszone Token and TokenVesting 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 Tankwarszone Token and TokenVesting Smart Contracts

Tank Wars Zone is an exciting and visually refreshing action game built on blockchain technology. With various gameplays, you can play solo, or gather your friends, form a team, and battle with others while being able to earn money just by playing or staking your NFTs.

WBOND Token is an ERC20 token that Tank Wars Zone players can use in the game.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of Tankwarszone Token and TokenVesting Smart Contracts. It was conducted on commit 6d9ea6bc032fce3d7129fbad0b9d839a449b117f from git repository *https://github.com/tankwars-zone/tank-contract/*.

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 Nov 26, 2021 and a total effort of 3 working days was dedicated to identifying and documenting security issues in the code base of the Tankwarszone Token and TokenVesting Smart Contracts.

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

FILE	SHA256 SUM
TokenVesting.sol	653de0af36881d6330db2359510a4f97c6dc277dbceb714ad91b8cf5dc29fab2
WBOND.sol	4278491c7798f67566f9496552544dadabdf5c684d3e4962c7ee3ca453e476f5

2.2. Contract code

The Tankwarszone Token and TokenVesting Smart Contracts was written in Solidity language, with the required version to be ^0.8.0. The source code was written based on OpenZeppelin's library.

The provided source codes consist of two contracts which inherit some contracts from OpenZeppelin.

2.2.1. WBOND token contract

WBOND 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. TokenVesting contract

With the vesting contract, the token distribution process can be summarized as below:

- Before the start time, all tokens are locked in the vesting contract without TGE.
- Once the start time is reached, all the tokens will be unlocked at the end of each cliff duration.

2.3. Findings

During the audit process, the audit team found no vulnerability in the given version of Tankwarszone Token and TokenVesting Smart Contracts.

2.4. Additional notes and recommendations

2.4.1. WBond.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 Tankwarszone Token and TokenVesting 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. TokenVesting.sol - Missing check length of array in lockToken function INFORMATIVE

In the lockToken function, the function uses amounts parameter but noncheck the length of this array variable. Therefore, the function can access an index which not inbound of the array causes an error in the function.

```
 92  function lockToken(uint256 poolId, address[] memory accounts, uint25…
     6[] memory amounts)
 93          public
 94          onlyOwner
 95          poolExist(poolId)
 96      {
 97          uint256 length = accounts.length;
 98
 99          uint256 total = 0;
100
101          for (uint256 i = 0; i < length; i++) {
102              address account = accounts[i];
103              uint256 amount = amounts[i];
104
105              require(account != address(0), "TokenVesting: address is…
      invalid");
106
107              require(amount > 0, "TokenVesting: amount is invalid");
```

Snippet 1. TokenVesting.sol Missing check array length in `lockToken` function

RECOMMENDATION

We suggest changing the function like the below code:

```
92  function lockToken(uint256 poolId, address[] memory accounts, uint25…
    6[] memory amounts)
93          public
94          onlyOwner
95          poolExist(poolId)
96      {
97          uint256 length = accounts.length;
98          uint256 total = 0;
99          require(length > 0 && length == amounts.length, "LockToken: …
    array length is invalid");
100         for (uint256 i = 0; i < length; i++) {
101             address account = accounts[i];
102             uint256 amount = amounts[i];
103
104             require(account != address(0), "TokenVesting: address is…
     invalid");
105
106             require(amount > 0, "TokenVesting: amount is invalid");
```

Snippet 2. TokenVesting.sol Recommend fixing in `lockToken` function

APPENDIX

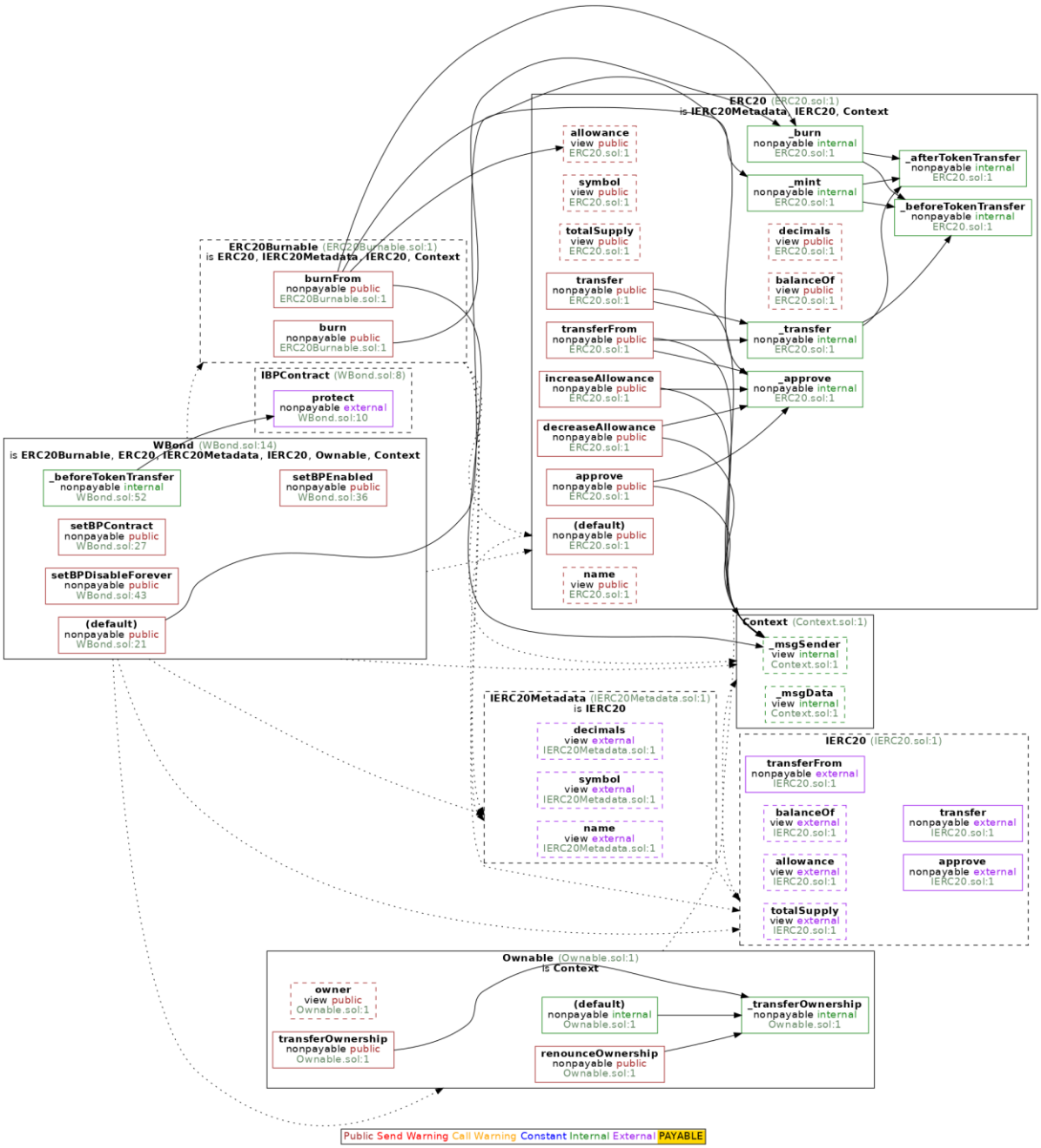

Image 1. Tankwarszone Token Smart contract call graph

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*2021-12-02*	Public Report	Verichains Lab

Table 2. Report versions history