

HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: WhiteBIT
Date: 9th, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed – upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for WhiteBIT.
Approved By	Evgeniy Bezuglyi SC Department Head at Hacken OU
Type	TRC20 token
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://whitebit.com
Timeline	18.04.2022 - 19.08.2022
Changelog	22.04.2022 - Initial Review 03.05.2022 - Second Review 06.05.2022 - Third Review 19.08.2022 - Fourth Review



Table of contents

Introduction	4
Scope	4
Severity Definitions	6
Executive Summary	7
Checked Items	8
System Overview	11
Findings	12
Disclaimers	14

Introduction

Hacken OÜ (Consultant) was contracted by WhiteBIT (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:

<https://github.com/whitebit-exchange/wbt-token>

Commit:

2ab92561d118bcf0801adeab5cddea86d813b61b

Technical Documentation: Yes

JS tests: Yes

Contracts:

BlackList.sol
TRC20Detailed.sol
TRC20.sol
ITRC20.sol
Ownable.sol
Pausable.sol
SafeMath.sol
WbtToken.sol

Second review scope

Repository:

<https://github.com/whitebit-exchange/wbt-token>

Commit:

6af16fadf6236648943cc55c433ce62c9016e273

Technical Documentation: Yes

JS tests: Yes

Contracts:

BlackList.sol
TRC20Detailed.sol
TRC20.sol
ITRC20.sol
Ownable.sol
Pausable.sol
SafeMath.sol
WbtToken.sol

Third review scope

Repository:

<https://github.com/whitebit-exchange/wbt-token>

Commit:

4f59525800442377fc106ddb44543f8537d6760b

Technical Documentation: Yes

JS tests: Yes



Contracts:

BlackList.sol
ERC20.sol
ERC20Detailed.sol
IERC20.sol
Ownable.sol
Pausable.sol
WbtToken.sol

Fourth review scope

Repository:

<https://github.com/whitebit-exchange/wbt-token>

Commit:

4d021a2029afdb3d4c22c0989c697c0469640bf9

Technical Documentation: Yes

JS tests: Yes

Contracts:

BlackList.sol
ERC20.sol
ERC20Detailed.sol
IERC20.sol
Ownable.sol
Pausable.sol
WBT.sol (Renamed from WbtToken.sol)

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Executive Summary

The score measurements details can be found in the corresponding section of the [methodology](#).

Documentation quality

The Customer provided superficial functional requirements and no technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **7** out of **10**. The limited number of unit tests provided.

As a result of the second review, CodeQuality score is changed to **10** out of **10**. Added good test coverage.

Architecture quality

The architecture quality score is **10** out of **10**. Code is well-structured and easy-readable.

Security score

As a result of the audit, security engineers found **1** critical, **0** high, **0** medium, and **5** low severity issues. The security score is **0** out of **10**.

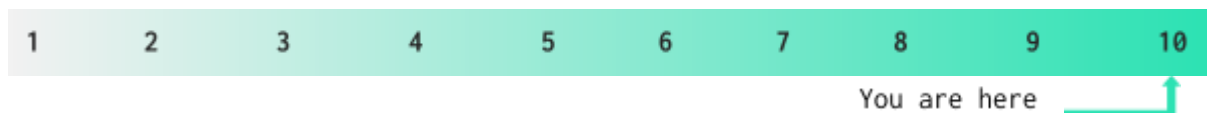
As a result of the second review, security engineers found **1** critical severity issue. **1** critical and **5** low severity issues from the previous revision were fixed. As a result, the code contains **1** critical severity issue. The security score is **0** out of **10**.

As a result of the third review, security engineers found no new issues. **1** critical severity issue from the previous revision was fixed. As a result, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10.0**



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Not Relevant
Check-Effect-I interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes.	Passed
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Leve1-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Custom	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	Custom	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed

Gas Limit and Loops	Custom	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block gas limit.	Passed
Style guide violation	Custom	Style guides and best practices should be followed.	Passed
Requirements Compliance	Custom	The code should be compliant with the requirements provided by the Customer.	Passed
Repository Consistency	Custom	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	Custom	The code should be covered with unit tests. Tests coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed



System Overview

WhiteBIT is Europe's largest international centralized crypto-to-fiat exchange with over 2 million registered users and a team of 350+ members that meet all KYC and AML requirements.

- Token – simple TRC-20 token that mints nothing after the deployment. Additional minting is allowed. The token has the ability to add addresses to the black list, which will stop all operations with the address. For blacklisted addresses, it has the ability to destroy funds.

It has the following attributes:

- Name: WhiteBIT WBT
- Symbol: WBT
- Decimals: 8
- Total supply: 400m (100m for TRC network)

Privileged roles

- The owner of the WBT contract can add or remove addresses from the blacklist to lock funds.
- The owner of the WBT contract can destroy funds for any blacklisted address.
- The owner of the WBT contract has the ability to burn tokens.
- The owner of the WBT contract can pause the contract, so all transfers would be stopped.
- The owner of the WBT contract can mint tokens without any limits.

Findings

■■■■ Critical

1. Mint is not limited.

The mint function is not limited, and any amount of tokens could be minted by the owner. It breaks the requirements to have a limit of 400 million tokens.

Recommendation: Restrict minting according to specification.

Status: Fixed (6af16fadf6236648943cc55c433ce62c9016e273)

2. Incorrect TRC20 interface.

Incorrect return values for TRC20 functions. A contract compiled with Solidity > 0.4.22 interacting with these functions will fail to execute them, as the return value is missing.

Token.transfer does not return a boolean. Bob deploys the token. Alice creates a contract that interacts with it but assumes a correct TRC20 interface implementation. Alice's contract is unable to interact with Bob's contract.

Recommendation: Set the appropriate return values and types for the defined TRC20 functions.

Status: Fixed (4f59525800442377fc106ddb44543f8537d6760b)

■■■ High

No high severity issues were found.

■■ Medium

No medium severity issues were found.

■ Low

1. Variable shadowing.

Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B, which has a state variable x defined. This would result in two separate versions of x, accessed from contract A and the other from contract B. In more complex contract systems, this condition could go unnoticed and subsequently lead to security issues.

Contract: TRC20.sol

Functions: TRC20.balanceOf(address owner) -> Ownable.owner(),

TRC20.allowance(address owner, address spender) -> Ownable.owner()

Recommendation: Consider renaming function argument.

Status: Fixed (6af16fadf6236648943cc55c433ce62c9016e273)

2. Boolean equality.

Boolean constants can be used directly and do not need to be compared to true or false.

Contract: TRC20.sol

Function: _transfer

Recommendation: Consider using “whenNotPaused” modifier instead.

Status: Fixed (6af16fadf6236648943cc55c433ce62c9016e273)

3. The public function could be declared external.

Public functions that are never called by the contract should be declared external to save Gas.

Contracts: Ownable.sol, Pausable.sol, TRC20.sol, TRC20Detailed.sol, Migrations.sol

Functions: owner, renounceOwnership, transferOwnership, pause, unpause, burn, destroyBlackFunds, name, symbol, decimals, setCompleted, upgrade

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (6af16fadf6236648943cc55c433ce62c9016e273)

4. Zero address is allowed.

“msg.sender” does not check if it is a zero address, which should be checked.

Contracts: TRC20.sol

Functions: approve, increaseAllowance, decreaseAllowance

Recommendation: Add check for zero address for msg.sender

Status: Fixed (6af16fadf6236648943cc55c433ce62c9016e273)

5. DestroyBlackFunds event has an incorrect burned amount value.

DestroyedBlackFund emits an event with the value from user balances “_balances[_blackListedUser]” which was reset to zero a step before.

Contract: TRC20.sol

Functions: destroyBlackFunds

Recommendation: Use “dirtyFunds” variable instead of “_balances[_blackListedUser]”

Status: Fixed (6af16fadf6236648943cc55c433ce62c9016e273)



Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.