SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: WhiteBIT
Date: AUGUST 31st, 2022

This report 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 can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for WhiteBIT
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Escrow
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://whitebit.com/
Timeline	21.06.2022 - 31.08.2022
Changelog	24.06.2022 - Initial Review 13.07.2022 - Second Review 31.08.2022 - Third Review

Table of contents

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:
5dac856aa3f596934d34938421ea32ca8b1b8d3c

Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Type: Technical description
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Type: Functional requirements
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/Escrow.sol
SHA3: aeef910255404b098ebfdb24d96d8a3cb25341eb9f76cc0d4587b59e7ce1ca5b

File: ./contracts/LimitedSetup.sol
SHA3: b124646af0b1b9732e602d496f44e6c7616efe7986590e0c0765fd922709f66b

File: ./contracts/Ownable.sol
SHA3: 360722ca30d8a7410de149ae9a02c9b7ab112081f6f4168a9ea3df1cd8c6872a

Second review scope

Repository:
https://github.com/whitebit-exchange/wbt-token
Commit:
a0116e1c8361301481e754878b06c4a97f0ed90f

Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Type: Technical description
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Type: Functional requirements
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Integration and Unit Tests: Yes

Contracts:

File: ./contracts/Escrow.sol
SHA3: 12eabf25bf645135009ad1a0b979b82ba52000ee00de8a80ea0a45521a1b17ce

File: ./contracts/LimitedSetup.sol
SHA3: b124646af0b1b9732e602d496f44e6c7616efe7986590e0c0765fd922709f66b

File: ./contracts/Ownable.sol
SHA3: d8dfc584f26c82e845986f630974fab22336ba974792b7a3392df974f2147eef

Third review scope

Repository:

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

Commit:

bec0cc57dfb5fba31decda33e524eb0b0ecef12d

Technical Documentation:

Type: Whitepaper (partial functional requirements provided)
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Type: Technical description
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Type: Functional requirements
https://drive.google.com/file/d/1RjWJ-9QGHZZS3RrJjUmZDNB-VTfltPBn/view

Integration and Unit Tests: Yes

Contracts:

File: ./contracts/Escrow.sol
SHA3: e45293f5fedb2418b08c6549cc5837d1834355235bbd6738569fded130d142a8

File: ./contracts/LimitedSetup.sol
SHA3: b124646af0b1b9732e602d496f44e6c7616efe7986590e0c0765fd922709f66b

File: ./contracts/Ownable.sol
SHA3: d8dfc584f26c82e845986f630974fab22336ba974792b7a3392df974f2147eef

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 measurement details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided superficial functional and technical requirements. The total Documentation Quality score is **7** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**. There are a lot of negative and positive cases.

Architecture quality

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

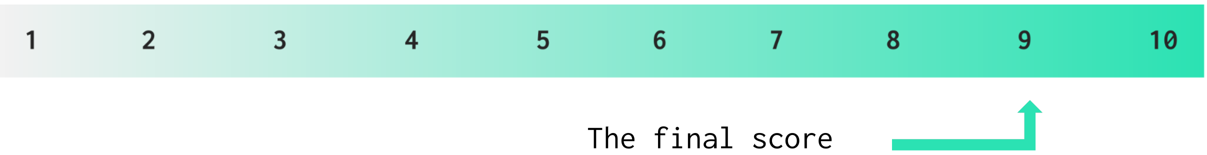
Security score

As a result of the third audit, security engineers found **1** medium and **5** low severity issues. The security score is **9** 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: **9.0**.



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.	Failed
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.	Passed
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 self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
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.	Not Relevant
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	SWC-116	Block numbers should not be used for time	Passed

a proxy for time		calculations.	
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	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 or be predictable.	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-Level-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.	Not Relevant
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.	Failed
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
Environment	**Custom**	The project should contain a configured	Passed

Consistency		development environment with a comprehensive description of how to compile, build and deploy the code.	
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	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.

- WhiteBIT Token – simple ERC-20 token.
  It has the following attributes:
  - Name: WhiteBIT WBT
  - Symbol: WBT
  - Decimals: 8
  - Total supply: 400m (100m for TRC network)
- *Escrow* – smart contract that holds 200m (WBT) ERC20 tokens for distribution for 3 years according to the distribution schedule. Escrow contract includes an 8 weeks setup period allowing to set final token distribution schedule.

Privileged roles

- The owner of the *Escrow* contract can transfer ownership to another address. The owner can leave the contract without an owner and remove functionality that is only available to the owner.
- The owner of the Escrow contract can add a new vesting entry at a given time and quantity to an account’s schedule. The owner of the Escrow contract can destroy the vesting information associated with an account. All these changes the owner can do only during the setup period. ‘Setup’ period is 8 weeks long. The length of the period is set when the contract is deployed and cannot be changed later.

Risks

- In case of an admin keys leak, an attacker can transfer ownership to another address or remove the owner of the contract at all. If the ‘setup’ period does not end yet, then the attacker can destroy the vesting information associated with any account. During the ‘setup’ period, attackers can add their accounts to the vesting schedule. If the ‘setup’ period does end, the attacker cannot do anything to steal funds or do something harmful for Escrow smart contract.

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

1. **Unchecked call return value.**
   The return value of a message call should be checked.

   The return value of a transfer call in the ‘vest’ function is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behavior in the subsequent program logic.

   File: ./contracts/Escrow.sol

   Contract: Escrow.sol

   Function: vest

   Recommendation: Consider the case when the transfer call could fail and check the return value.

   Status: Fixed

2. **Costly operations inside a loop.**
   Costly operations inside a loop might waste Gas, so optimizations are justified.

   The loop inside the addVestingSchedule is not optimized. addVestingSchedule calls addVestingEntry inside the loop, and addVestingEntry function will change totalVestedBalance state many times. Making this computation over and over again will be costly in terms of Gas.

   This can lead to high Gas consumption.

   File: ./contracts/Escrow.sol

   Contract: Escrow.sol

   Function: addVestingSchedule

   Recommendation: Declare local variable _totalVestedBalance and update it inside the loop. When the loop ends, updating the totalVestedBalance contract’s state variable only one time will reduce the transaction execution costs.

Status: Reported

Low

1. **Gas limits and loops.**
Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should be no cases when execution fails due to the block Gas limit.

Function vest() iterates through all vestingSchedules’s items. This approach will increase the costs of transaction execution. There are some limitations (20 records per address). However, the 20th ‘vest’ request will be much more expensive than the first ‘vest’ request.

File: ./contracts/Escrow.sol

Contract: Escrow.sol

Function: vest

Recommendation: Decrease the count of items in vestingSchedules state. It will reduce the costs of transaction execution.

Status: Reported

2. **Test coverage.**
Test coverage should be 100%, with negative and positive cases covered.

Function renounceOwnership() and transferOwnership(address) not covered with tests. No test cases for ‘Escrow’ function are covered when ‘setup’ period is ended. No test cases for Escrow's functions are covered with the incorrect owner’s address.

Files: ./contracts/Ownable.sol

./contracts/Escrow.sol

Contracts: Ownable.sol, Escrow.sol

Functions: renounceOwnership, transferOwnership, addVestingSchedule, purgeAccount, appendVestingEntry

Recommendation: Cover renounceOwnership and transferOwnership functions with negative and positive cases. Cover addVestingSchedule, purgeAccount, ‘appendVestingEntry functions with negative test cases when ‘setup’ period is ended. Cover addVestingSchedule, purgeAccount, appendVestingEntry functions with negative cases when the owner address is incorrect or deleted via renounceOwnership() function.

Status: Fixed

3. **Functions that can be declared external.**

In order to save Gas, public functions that are never called in the contract should be declared as external.

Files: ./contracts/Escrow.sol,

./contracts/Ownable.sol

Contracts: Escrow, Ownable

Functions: Escrow.balanceOf, Ownable.owner

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

Status: Reported

4. **Missing zero address validation.**

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

File: ./contracts/Escrow.sol

Contract: Escrow

Constructor variable: _token

Function variable: spender

Recommendation: Implement zero address checks.

Status: New

5. **Functions that can be declared external.**

In order to save Gas, public functions that are never called in the contract should be declared as external.

Files: ./contracts/Escrow.sol,

./contracts/Ownable.sol

Contracts: Escrow, Ownable

Functions: Escrow.balanceOf, Ownable.owner

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

Status: Reported

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.