HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ZOA
Date: April 26th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for ZOA.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC20 tokens
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	
Timeline	21.04.2022 - 26.04.2022
Changelog	23.04.2022 - Initial Review 26.04.2022 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by ZOA (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/zoa-game-p2e/zoa-token/tree/master
Commit:
ad3f7a10ae992ea152d464b9be6fa1ca2512b0f9
Technical Documentation: Yes
JS tests: Yes
(https://github.com/hknio/zoa-token-460173551/tree/master/test)
Contracts:
File: ./contracts/avax/ZOAToken.sol
SHA3: 5fb7da752286f24435768d66a4e894facc25ae05edd5ac38089de99b6feb9cb7

File: ./contracts/bsc/external/UniswapV2Library.sol
SHA3: df24119261a57aa5b7e4d1d0aafb0157b170ed57c6292a926cd23c70b9b6f26d

File: ./contracts/bsc/external/UniswapV3Library.sol
SHA3: 1a5798e8db90b80d966c402368069d18f57eb6906e0c4c8920c501122cfa739b

File: ./contracts/bsc/IPLPS.sol
SHA3: 6f40bde94a344eb73e74a0b45353fb7d51c862dade0bf1f7901a56c20569f564

File: ./contracts/bsc/UsingLiquidityProtectionService.sol
SHA3: 7e1a0a8505e7f075bc8911f3be3d9ec7d369d07128fa1c21c7c5da80ee9ed9ba

File: ./contracts/bsc/ZGOTokenBSC.sol
SHA3: 12b0837d8ffbd78227c0a6004f904b6dad41d367ff1b5a416af4263931443de8

File: ./contracts/bsc/ZOATokenBSC.sol
SHA3: c6045d0920d69fc8f0a6c739d3644da1bfc85b2910d23623ea5492ddb652ca38

File: ./contracts/lib/ERC20Blacklist.sol
SHA3: fcdc16c6b0570605b0bc36e38b9e3ecf4d91e91aa1c50e19c632dbdb53529891

Second review scope

Repository:
https://github.com/zoa-game-p2e/zoa-token/tree/master
Commit:
f2a38a0ddebe2c2c724023b5646d5700b3d40a4f
Technical Documentation: Yes
JS tests: Yes
(https://github.com/hknio/zoa-token-460173551/tree/master/test)
Contracts:
File: ./contracts/avax/ZOAToken.sol
SHA3: 5fb7da752286f24435768d66a4e894facc25ae05edd5ac38089de99b6feb9cb7

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

```
File: ./contracts/bsc/external/UniswapV2Library.sol
SHA3: df24119261a57aa5b7e4d1d0aafb0157b170ed57c6292a926cd23c70b9b6f26d

File: ./contracts/bsc/external/UniswapV3Library.sol
SHA3: 1a5798e8db90b80d966c402368069d18f57eb6906e0c4c8920c501122cfa739b

File: ./contracts/bsc/IPLPS.sol
SHA3: 6f40bde94a344eb73e74a0b45353fb7d51c862dade0bf1f7901a56c20569f564

File: ./contracts/bsc/UsingLiquidityProtectionService.sol
SHA3: 4465edd282614e9cc39cd1c1c16d941b66feca7a17e14c609c19eaf7e859552e

File: ./contracts/bsc/ZGOTokenBSC.sol
SHA3: 12b0837d8ffbd78227c0a6004f904b6dad41d367ff1b5a416af4263931443de8

File: ./contracts/bsc/ZOATokenBSC.sol
SHA3: c6045d0920d69fc8f0a6c739d3644da1bfc85b2910d23623ea5492ddb652ca38

File: ./contracts/lib/ERC20Blacklist.sol
SHA3: fcdc16c6b0570605b0bc36e38b9e3ecf4d91e91aa1c50e19c632dbdb53529891
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total Code Quality score is **5** out of **10**. The project has redundant code. Unit tests were provided.

Architecture quality

The architecture quality score is **5** out of **10**. The project has unoptimized inheritance logic.

Security score

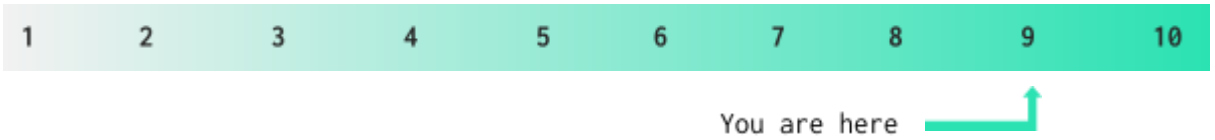
As a result of the audit, security engineers found **1** medium and **3** low severity issues. The security score is **10** out of **10**.

As a result of the second review, security engineers found no new issues. **1** medium issue from the previous revision was fixed and **1** low issue was mitigated. As a result, the code contains **2** low 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: **9**



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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-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.	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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	Not Relevant
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.	Failed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Failed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

ZOA is a project with a tokens contracts for the gaming ecosystem:

- ZOAToken — ERC-20 token that mints all initial supply to a deployer. Additional minting is not allowed. The contract is pausable, with blacklist functionality.
  It has the following attributes:
  - Name: Zone of Avoidance
  - Symbol: ZOA
  - Decimals: 18
  - Total supply: 1 000 000 000 tokens.
- ZGOTokenBSC — BEP-20 token with an unlimited supply, blacklist functionality, and LiquidityProtectionService checks.
  It has the following attributes:
  - Name: ZGO
  - Symbol: ZGO
  - Decimals: 18
  - Total supply: 0 tokens initially (unlimited minting).
- ZOATokenBSC — BEP-20 token that mints all initial supply to a deployer. Additional minting is not allowed. The contract is pausable, with blacklist functionality.
  It has the following attributes:
  - Name: ZOA
  - Symbol: ZOA
  - Decimals: 18
  - Total supply: 1 000 000 000 tokens.

Privileged roles

- The owner of the ZOAToken, ZOATokenBSC, and ZGOTokenBSC contracts may blacklist any account.
- The owner of the ZOAToken, ZOATokenBSC, and ZGOTokenBSC contracts may pause the contract.
- The owner of the ZOATokenBSC contract may unlimitedly mint any amount of tokens.
- The owner of the ZOATokenBSC contract may update the address of the LiquidityProtectionService contract.

Notes

- The liquidity protection service contract implementation is out of the audit scope.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found

High

No high severity issues were found.

Medium

1. **Unexpected state variable change**

   The abstract contract UsingLiquidityProtectionService has the function `revokeBlocked`, which sets the protection flags (e.g. `unProtected`, `unProtectedExtra`) to `false` after the execution. This behavior may revert changes after calls to `disableProtection` or `disableProtectionExtra` function.

   The state variables (e.g. `unProtected`, `unProtectedExtra`) may be changed unexpectedly, which significantly affects the contract behavior.

   Contracts: UsingLiquidityProtectionService.sol

   Function: revokeBlocked

   Recommendation: Rework the `revokeBlocked` function logic not to affect the state variables after the execution.

   Status: Fixed

Low

1. **Missing zero address validation**

   Address parameters are being used without checking against the possibility of 0x0 value in the `LiquidityProtection_setLiquidityProtectionService` function.

   This can lead to unwanted external calls to 0x0.

   Contracts: UsingLiquidityProtectionService.sol

   Function: LiquidityProtection_setLiquidityProtectionService

   Recommendation: Add `require` statement to check new address value.

   Status: Reported

2. **Missing event emitting**

   Events for critical state changes (e.g. update of LiquidityProtectionService contract address) should be emitted for tracking things off-chain.

   Contracts: UsingLiquidityProtectionService.sol

   Function: LiquidityProtection_setLiquidityProtectionService

Recommendation: Create and emit related event.

Status: Mitigated (The event is not emitted by design, to not trigger any malicious bot)

3. **Redundant functions**

The abstract contract UsingLiquidityProtectionService has the redundant functions which wrap the functions and mirror their return values.

Contracts: UsingLiquidityProtectionService.sol

Function: ProtectionSwitch_manual, ProtectionSwitch_manual_extra

Recommendation: Remove redundant wrapper functions.

Status: Reported

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.