HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Hyksos
Date: June 8th, 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 Hyksos
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC721; DeFi Loans
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.hyksos.fi/
Timeline	13.05.2022 - 08.06.2022
Changelog	19.05.2022 - Initial Review 08.06.2022 - 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 Hyksos (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/hyksos-fi/hyksos-contracts/tree/audit
Commit: fc4360879923da3f54378604a7ab003ab6bda702
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://hyksos.gitbook.io/hyksos

Type: Pitch Deck (superficial functional requirements provided)
Link:https://www.canva.com/design/DAE9NOHrmK8/vFCcP-ahHi54p-wcGmSfsw/view?utm_content=DAE9NOHrmK8&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink#6
JS tests: Yes
(https://github.com/hyksos-fi/hyksos-contracts/tree/master/test)
Contracts:
File: ./contracts/AutoCompound.sol
SHA3: 857b127774f1a710ae6c9dfc4feaa01e33d6eac07dfe002658ccbad2d610f416

File: ./contracts/DepositQueue.sol
SHA3: b99f57974c551b56355d3363165b8056dfd9b9a561f1b16841c63ab26a2d7a61

File: ./contracts/HyksosBase.sol
SHA3: d006fa1f4eda82be3f2e7e3fef8f9120d2fd2f3e45c6538941c61ff0b070edbf

File: ./contracts/HyksosCyberkongz.sol
SHA3: 8ea8af57a4b57a1eec38f54f20858d951e89c6fef1cd58e15de4111234654f07

File: ./contracts/HyksosEtherorcs.sol
SHA3: b9fcf065bc2462f466d3271b8ac34715e6c38dde6fdea7b951532d3560a43672

File: ./contracts/IHyksos.sol
SHA3: a5770e24c89f33699cfed24e15947d9c9b5b422613fc06ddb92fb6fc69032084

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

Documentation quality

The Customer provided functional requirements in the whitepaper and project repository. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **9** out of **10**. Code violates the order of functions defined in the style guide.

Architecture quality

The architecture quality score is **10** out of **10**. The code follows best practices.

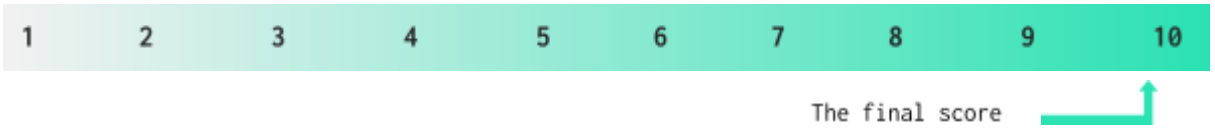
Security score

As a result of the audit, the code contains **0** 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: **9.9**



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.	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 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
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.	Not Relevant
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.	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.	Passed

Style guide violation	**Custom**	Style guides and best practices should be followed.	Failed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Not Relevant
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. 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

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

System Overview

Hyksos is a mixed-purpose system with the following contracts:

- *AutoCompound* - a contract that handles auto compound for an address.
- *DepositQueue* - a contract to track users` deposits with functions to work with the deposits array.
- *HyksosBase* - an abstract contract of an NFT-backed loan platform that can be extended by a specific implementation that includes NFT features. This contract distributes and transfers rewards *to shareholders accordingly to their stake and compounding strategy.*
- *HyksosCyberkongz* - a contract to deposit ERC20 tokens (IBananas) and NFT (IKongs) and withdraw deposited funds with rewards distribution.
- *HyksosCyberorcs* - a contract to deposit ERC20 tokens (Zug) and NFT (IOrcs) and withdraw deposited funds with rewards distribution.
- *IHyksos* - an interface for HyksosBase contract.

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. **Unchecked call return value**

   The return value of a message call 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.

   This can lead to funds losses

   Contract: HyksosCyberkongz.sol

   Functions: payErc20, withdrawErc20, depositErc20

   Recommendation: By choosing low-level call methods, make sure to handle the possibility that the call will fail by checking the return value.

   Status: Fixed(fc43608)

2. **Costly operations inside a loop**

   Loops are not optimized. Functions do not feature local state variables that save Gas.

   This can lead to high Gas losses.

   Contracts: HyksosBase.sol

   Function: distributeRewards, withdrawNftAndShareRewardEqually, withdrawNftAndRewardClaimant, withdrawNftAndRewardOwner, selectShareholders.

   Recommendation: Cache array length, use *for* loop instead of *while.*

   Status: Fixed(fc43608)

Low

1. **Floating Pragma.**

   Contracts files use floating pragma ^0.8.0

   Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

   Contracts: AutoCompound, DepositQueue, HyksosBase, HyksosCyberkongz, HyksosEtherOrcs, IHyksos

Function: -

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed(fc43608)

2. **State Variable Default Visibility**

   The explicit visibility makes it easier to catch incorrect assumptions about who can access the variable.

   Contracts: AutoCompound, DepositQueue, HyksosBase, HyksosCyberkongz, HyksosEtherOrcs

   Function: -

   Recommendation: Variables can be specified as *public*, *internal,* or *private*. Explicitly define visibility for all state variables.

   Status: Fixed(fc43608)

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.