HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Hypercycle
Date: 19 September, 2023

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 Hypercycle
Approved By	Arda Usman | Lead Solidity SC Auditor at Hacken OÜ
Tags	Fungible Token; Non-fungible Token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.hypercycle.ai/
Changelog	22.08.2023 - Initial Review 31.08.2023 - Second Review 19.09.2023 - Third 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 Hypercycle (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.

System Overview

HypercycleV2 is a crowd protocol with the following contracts:

- *CrowdFundHYPCPoolV2* — allows users to gather ERC20 HyPC tokens together to swap for ERC721 c_HyPC.

- c_HyPC can be used to back a license in the HyperCycle ecosystem.
- Since a high volume of HyPC is required for a swap, a pooling contract is useful for users wanting to have HyPC back their licenses.
- A license holder can create a proposal in the pool for 1 c_HyPC to back their license. They add HyPC as collateral for this lean, which will be used as interest payments for the users that provide HyPC for the proposal.
- A license holder can create a proposal for multiple c_HyPC instead of just one.

Privileged roles

- Contract Owner: change pool fees.
- Proposal Owner: change proposal assignments, transfer ownership, cancel proposals.
- Deposit Owner: transfer deposits, update deposits, withdraw deposits.

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 scoring methodology.

Documentation quality

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

- Functional requirements are detailed.
- Run instructions are provided.
- NatSpec is sufficient.

Code quality

The total Code Quality score is **10** out of **10**.

- An updated Solidity version is used.
- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

Security score

As a result of the audit, 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**. The system users should acknowledge all the risks summed up in the risks section of the report.

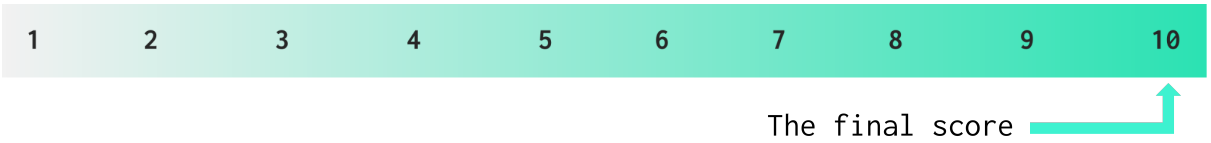

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
22 August 2023	7	3	1	0
31 August 2023	0	1	0	0
19 September 2023	0	0	0	0

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

Risks

- The audited contract interacts with other contracts that are out-of-scope, and thus its logic cannot be validated: CHYPC.sol, HyperCycleToken.sol, HyperCycleSwap.sol.
- Fees do not have constraints and expected behavior on proposal creation relies on the data inputted by the frontend. In the case the fee is too high and the frontend blindly inputs the fee, the amount transferred can be unexpected by the final user. Before calling the createProposal() method, the user should double check the fee amount.

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

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	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	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

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

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	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	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

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

Flashloan Attack	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. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	
Token Supply Manipulation	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	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	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	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	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

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

Findings

■■■■ Critical

■■■ High

H01. Requirements Violation

Impact	High
Likelihood	Medium

The poolFee is not set during the contract deployment (constructor), but manually afterward.

As a consequence, users can create proposals for free during the time span between the contract deployment and the call to setPoolFee().

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: setPoolFee().

Recommendation: Consider calling setPoolFee() in the constructor().

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

■■ Medium

M01. Undocumented Functionality

Impact	Medium
Likelihood	Medium

The assign() function calls as well as the assignmentStrings stored are empty strings.

There is no apparent reason and no documentation about this fact.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: redeemTokens(), swapTokens().

Recommendation: Consider providing additional documentation about why the strings are empty or update the code.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

M02. Inefficient Gas Model

Impact	Medium

Likelihood	Medium

Most functions in the contract use storage pointers and/or access storage variables multiple times within the body function, consuming a lot of Gas unnecessarily.

Instead, it is recommended to use memory pointers when reading from such variables or creating a memory variable when it is going to be read multiple times.

However, for those variables that are written into storage, this approach should not be used.

For the pointer case, taking the example of withdrawDeposit(): the storage pointer ContractProposal storage proposalData should instead be defined as ContractProposal memory proposalData. But, when updating depositAmount, instead of proposalData.depositedAmount, proposals[proposalIndex].depositedAmount should be used.

For the memory case, taking the same example of withdrawDeposit(): the storage variable userDeposits[msg.sender][depositIndex] is accessed multiple times. Instead, creating a new memory variable defined as userDeposits[msg.sender][depositIndex] to be used within the body function would save Gas.

Another case in withdrawDeposit() is the declaration of the memory variable amount = userDeposits[msg.sender][depositIndex].amount, which can be used in the previous line of code: proposalData.depositedAmount -= amount.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: createDeposit(), transferDeposit(), swapTokens(), redeemTokens(), withdrawDeposit(), updateDeposit(), completeProposal(), finishProposal(), changeAssignment().

Recommendation: Consider using memory pointers and new memory variables to decrease the Gas cost of the reported functions. Pass storage variables to memory, apply changes and commit the changes to the storage after all changes.

Found in: c685e21

Status: Fixed (Revised commit: 915071d)

M03. NatSpec Contradiction

Impact	Low
Likelihood	High

The implementation of some functions and events does not adhere to its NatSpec specification.

These inconsistencies can cause confusion and make it harder for auditors and developers to understand the code. Additionally, even small inconsistencies can accumulate over time and make the codebase harder to maintain.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol:

- AssignmentChanged: missing parameter
- TokensSwapped: wrong NatSpec format
- PoolFeeSet: wrong poolFee parameter description
- changeAssignment(): missing assignmentString parameter

Recommendation: Fix the NatSpec contradictions.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

Low

L01. Unused Variable

Impact	Low
Likelihood	Medium

The state variable _1_MONTH is never used.

Unused variables should be removed from the contract.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: _1_MONTH

Recommendation: It is recommended to remove the reported variable.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

L02. Redundant View Functions

Impact	Low
Likelihood	Medium

The following functions return state variables that are already public, and thus can already be read without creating additional functions.

Redundant code should be removed from the contracts for simplification and decrease of the contract deployment Gas cost.

Path:./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: getDeposit(), getProposal().

Recommendation: Consider removing redundant functions.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

L03. Redundant Code

Impact	Low
Likelihood	Medium

In both transferDeposit() and withdrawDeposit(), the code *delete userDeposits[msg.sender][depositIndex]* is redundant since the next line of code will already overwrite the value.

Path:./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: transferDeposit(), withdrawDeposit().

Recommendation: Consider removing redundant code.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

L04. Missing Zero Address Validation

Impact	Low
Likelihood	Low

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

This can lead to unwanted external calls to 0x0.

Path:./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: transferDeposit(), transferProposal().

Recommendation: Implement zero address checks.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

L05. Undocumented Parameter

Impact	Low
Likelihood	Low

The functions createProposal() and swapTokens() use the hardcoded value 524288 with no context or additional information.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: createProposal(), swapTokens().

Recommendation: Consider using a constant state variable to define the origin of 524288 instead of hardcoding it.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

L06. Redundant Code

Impact	Low
Likelihood	Medium

The modifier `validIndex()` requires the `proposalsArray` parameter and that parameter is not useful as every occurrence of calls for `validIndex()` are passing the contract state variable `proposals`.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: validIndex()

Recommendation: Consider removing redundant code. Remove `proposalsArray` parameter and read from contract storage directly in the modifier scope.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

L07. Best Practice Violation - Checks-Effects-Interactions Pattern

Impact	High
Likelihood	Low

State variables are updated after the external calls to the token contract.

As explained in Solidity Security Considerations, it is best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: swapTokens() → tokenId = SwapContract.nfts(0).

Recommendation: Follow the checks-effects-interactions pattern when interacting with external contracts.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

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

Informational

I01. Missing Events for Critical Value Updates

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: startProposal(), completeProposal().

Recommendation: Consider emitting *events* in said functions.

Found in: c685e21

Status: Fixed (Revised commit: 5bb42fa)

I02. Solidity Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, view and pure functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all functions are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: createProposal(), swapTokens().

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Providing comprehensive NatSpec annotations for functions and following Solidity's naming conventions further enrich the quality of the code.

Found in: c685e21

Status: Fixed (Revised commit: 915071d)

I03. Usage of Magic Number

The functions createProposal() and swapTokens() use the hardcoded value 524288 without using a descriptive constant that can be reusable and easier to understand.

Path: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol: createProposal(), swapTokens().

Recommendation: Define and use a constant state variable to define the origin of 524288 instead of hardcoding it on the functions scope.

Found in: 5bb42fa

Status: Fixed (Revised commit: 915071d)

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 based on best industry practices at the time of the writing 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 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, the Consultant cannot guarantee the explicit security of the audited smart contracts.

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

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

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

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

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

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/hypercycle-development/hypercycle-contracts
Commit	c685e21
Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol SHA3: 91370f49e8e8f93c60963ceca239c54ed8dd609ef9cb5cd38e75fb233d39a54e

Second review scope

Repository	https://github.com/hypercycle-development/hypercycle-contracts
Commit	5bb42fa
Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/ethereum/core/CrowdFundHYPCPoolV2.sol SHA3: 168141ac469eca4e1691cf4633704dcd0d907642967d38edd99a9e97a19f2aee

Third review scope

Repository	https://github.com/hypercycle-development/hypercycle-contracts
Commit	915071d
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/ethereum/core/CrowdFundHYPCPoolV2.sol SHA3: 898ee50c758e66b812768697574f7cedb55a9acf25861d339c4afd854342d3a1