SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Hypercycle
Date: 30 May, 2023

Document

Name	Smart Contract Code Review and Security Analysis Report for Hypercycle
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC721; Marketplace; Staking;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.hypercycle.ai/
Changelog	11.05.2023 - Initial Review 29.05.2023 - Second Review 30.05.2023 - Third Review

Table of contents

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

Hypercycle is a staking protocol with the following contracts:

- *CHYPC* — an ERC721 NFT collection that includes the functionality to mint, burn, and assign custom data to NFT tokens. The contract interacts with two other contracts, HYPCSwap and HYPC, and supports the management of token ownership and data assignment, while ensuring proper initialization and contract interactions.
- *CrowdFundPoolHYPC* — a crowdfunding pool for Hypercycle (HYPC) tokens, allowing users to create proposals, deposit funds, and earn interest over time. It manages the lifecycle of proposals and the interactions between users, such as depositing, withdrawing, and updating interest earnings, as well as handling token swaps and NFT assignments.
- *HYPCSwap* - a smart contract used to swap HYPC tokens for a CHYPC NFT and vice versa. It manages the token and NFT balances while maintaining an up-to-date record of NFTs available for swapping.
- *ICHYPC* - an interface for the *CHYPC* smart contract.
- *IHYPC* - an interface for the *HYPC* ERC20 token smart contract.
- *IHYPCSwap* - an interface for the *HYPCSwap* smart contract.

Privileged roles

- CHYPCNFT.sol :
  - Contract Owner :
    - Can init the contract.
    - Can mint tokens.
  - Token Owner :
    - Can burn the token.
    - Can set an assignment to the token.
- HyperCycleSwap.sol :
  - Token Contract (CHYPCAddress) :
    - Can add an NFT.
- CrowdFundHYPCPool.sol :
  - Proposal owner :
    - Can cancel a proposal.
    - Can finish the proposal.
    - Can change the assignment.

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 present, but only at a high-level.
  - Functional requirements for how the system should work are provided.
  - The documentation has a detailed description of the math calculations.
- Technical description is provided.
  - Run instructions are provided.
  - Technical specification is provided.
  - NatSpec is sufficient.

Code quality

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

- The development environment was configured.
- The code is well-designed and follows best practices.

Test coverage

Code coverage of the project is **100.0%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Tests are not configured to run in a local environment.

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**.

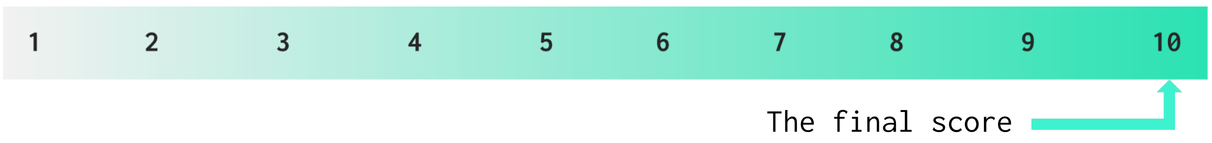

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
11 May 2023	8	4	3	1
29 May 2023	2	0	0	0
30 May 2023	0	0	0	0

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.	Not Relevant	
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	

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.	Not Relevant	
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	

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.	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.	Passed	
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	

Findings

■■■■ Critical

C01. Invalid Validation; Funds Lock; Data Consistency

Impact	High
Likelihood	High

In the *withdrawDeposit()* function, the removal of the deposit element from the user deposits array is done incorrectly.

The validation *if (userDeposits[msg.sender].length == 1 || depositIndex == userDeposits[msg.sender].length - 1)* should check if the element to remove is not the first or the last element of the array, and then perform the array reordering if this is the case.

But currently it reorders when the element is first or last.

This leads to a situation where, when an array element with an index from the middle of the array is used, the last element of the array is removed instead of the desired element.

As a consequence of the incorrect element removal, the funds are locked and the contract data is corrupted.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : withdrawDeposit()

Recommendation: Fix the validation in the if statement to correctly check for the element in the middle of the array in order to correctly reorder the array before the last element is popped.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

■■■ High

H01. Requirements Violation

Impact	Medium
Likelihood	High

According to the documentation provided by the customer, only a manager should be able to create the proposal: *A manager creates a proposal, providing backingFunds for interest.*

The code does not match the documentation; any external address can call the *createProposal()* function.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : createProposal()

Recommendation: Adjust the documentation to reflect the code, or adjust the code to reflect the documentation.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

H02. Undocumented Behaviour

Impact	Medium
Likelihood	High

Many hard-coded numbers and formulas are used without any explanation:

```
startNumber = 67108864
```

```
uint interestRate = (backingFunds * 26000000000) /
(requiredFunds * periods)
```

```
uint interestChange = (periods * deposit.amount *
proposalData.interestRate) / 26000000000
```

The code should not contain undocumented functionality.

Paths:
./contracts/ethereum/core/CHYPC.sol : initContract()
./contracts/ethereum/core/CrowdFundHYPCPool.sol : createProposal(), updateDeposit()

Recommendation: Provide documentation about numbers and calculations.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

H03. Race Condition; Undocumented Behaviour

Impact	High
Likelihood	Medium

In the createDeposit() function, the user who fills the pool is penalized during the deposit process. This user will need to pay additional Gas to fulfill the swap and the assignment.

Anyone can frontrun him to avoid paying this extra Gas fee, creating a race condition.

In addition, his deposit will not be equal to what the user specified in the function parameter, forcing him to accept a bad trade if the reward from depositing a `new` amount is less than the additional transaction cost.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : createDeposit()

Recommendation: The *createDeposit()* function should only be used to accept deposits into a proposal. Consider implementing a new function, startProposal(), that can be called by anyone once deposits are full.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

Medium

M01. Unfinalized Code

Impact	Medium
Likelihood	Medium

The code contains code comments which show that the code has not been finalized and is not ready for production.

// Create tests for when user wants to withdraw diferent deposits

Even though the comment refers to the test suite, the underlying untested code contains a critical vulnerability, which would have been detected if the code had been finalized.

Path:
./contracts/ethereum/core/CrowdFundHYPCPool.sol : withdrawDeposit()

Recommendation: Implement tests based on this comment and remove it when implemented.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

M02. Inefficient Gas Model

Impact	Medium
Likelihood	Medium

When the *validIndex()* modifier is used, the elements from the *proposals* array are copied into a memory function parameter, *ContractProposal[] memory proposalsArray*.

Elements from *proposalsArray* are not used within the modifier; only the length of the array is extracted.

As the copying operation into the memory consumes Gas, this copy causes an increase in the Gas cost for each element present in the array when applied to functions.

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

Recommendation: Use *storage* for the *proposalsArray* parameter to simply pass the reference and do not copy it to memory.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

M03. Inconsistent Data

Impact	Low
Likelihood	High

In the *createDeposit()* function, the event *DepositCreated* is emitted in the else case; however, the amount value is not correct; it should be *newAmount* instead of *amount*.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : createDeposit()

Recommendation: Pass the correct value to the event.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

M04. Race Condition; Undocumented Behaviour

Impact	Medium
Likelihood	Medium

In the *updateDeposit()* function, the user who will perform the action when the timestamp from *ContractProposal.term* has passed will have to pay extra Gas for calling the redeem() and assign() functions for the NFT from the proposal.

This leads to a situation where the user is penalized for his action.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : updateDeposit()

Recommendation: Implement a dedicated function, *completeProposal()*, to update the proposal state from STARTED to COMPLETED in the event that ContractProposal.term is reached.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

Low

L01. Redundant State Variable

Impact	Low
Likelihood	Low

In CrowdFundPoolHYPC.sol, the swap contract address is stored twice in two different variables : SwapContract and swapAddress.

In HYPCSwap.sol, the NFT token contract address is stored twice in two different variables : CHYPCAddress and HYPCNFT.

In CHYPC.sol, the swap contract address is stored twice in two different variables : HYPCSwapAddress and HYPCSwapContract.

Paths:
./contracts/ethereum/core/HYPCSwap.sol : CHYPCAddress
./contracts/ethereum/core/CHYPC.sol : HYPCSwapAddress
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : swapAddress

Recommendation: Store the value only once with the appropriate data type, and cast it to an address if needed.

Found in: e60234e

Status: Fixed (22ecf5413d10e2d5ef3b6153b65ca80d92027671)

L02. Use Of Hard-Coded Values

Impact	Medium
Likelihood	Low

Using hard-coded values in the computations and comparisons is not the best practice.

Paths:
./contracts/ethereum/core/HYPCSwap.sol : swap(), redeem()
./contracts/ethereum/core/CHYPC.sol : initContract()
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : createProposal()

Recommendation: Convert these variables into constants.

Found in: e60234e

Status: Mitigated (Hard-coded values are extensively described in the NatSpec comments.)

L03. Modification Of A Well-Known Contract

Impact	Low
Likelihood	Low

The *HYPCSwap* contract uses a well known reentrancy pattern, instead of importing existing code.

It makes the code less clear and readable.

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

Recommendation: Delete the modifications and use the OpenZeppelin ReentrancyGuard library.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

L04. Missing Zero Address Validation

Impact	Medium
Likelihood	Low

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

This can lead to unwanted external calls to 0x0.

Path:
./contracts/ethereum/core/CHYPC.sol : initContract()

Recommendation: Implement zero address checks.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

L05. Variable Shadowing

Impact	Medium
Likelihood	Low

In the *assign()* function, the variable *owner* is shadowing the *owner()* function from the *Ownable* contract.

The use of the owner local variable is redundant as the value from *ownerOf(tokenId)* can be used directly in the `require` comparison, and msg.sender in the emitted event.

Path:
./contracts/ethereum/core/CHYPC.sol : assign()

Recommendation: Rename or delete the related variable.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

L06. Missing Validation

Impact	Low
Likelihood	Medium

The getAssignment() function gets as a parameter the id of an ERC721 token, but does not check if these tokens are already minted.

Path:
./contracts/ethereum/core/CHYPC.sol : getAssignment()

Recommendation: Add a check *require(tokenId >= startNumber && tokenId < totalMinted, "Token not yet minted.");*.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

L07. Interfaces Mismatch

Impact	Medium
Likelihood	Low

IHYPCSwap.sol is supposed to represent HyperCycleSwap.sol but is not inherited by it.

ICHYPC.sol is supposed to represent CHYPCNFT.sol but is not inherited by it.

Paths:
./contracts/ethereum/core/CHYPCNFT.sol
./contracts/ethereum/core/HyperCycleSwap.sol

Recommendation: Contracts implementations should inherit from their interface definitions.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

L08. Missing Validation

Impact	Low
Likelihood	Medium

The *changeAssignment()* function should also have modifiers *validIndex* and *proposalOwner*, instead of checking the requirements in its body.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : changeAssignment()

Recommendation: Replace checks from the function body with already defined modifiers.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

Informational

I01. Boolean Equality

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

Paths:
./contracts/ethereum/core/HYPCSwap.sol : swap(), redeem()
./contracts/ethereum/core/CHYPC.sol : initContract(), burn(), mint(), assign()

Recommendation: Remove boolean equality.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I02. Style Guide Violation - Naming Mismatch

The names of the contracts should be equal to the file names.

Path:
./contracts/ethereum/core/HYPCSwap.sol
./contracts/ethereum/core/CHYPC.sol
./contracts/ethereum/core/CrowdFundPoolHYPC.sol;

Recommendation: Change filenames according to smart contracts naming.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I03. Immutable Variables

Compared to regular state variables, the Gas costs of constant and immutable variables are much lower. Immutable variables are evaluated once at construction time and their value is copied to all the places in the code where they are accessed.

Path:
./contracts/ethereum/core/HYPCSwap.sol : HYPCToken, HYPCNFT, CHYPCAddress

Recommendation: Declare variables mentioned as immutable to save gas on user operations.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I04. Contradiction - Revert Message

The revert message: *“Must be swap address”*, does not reflect reality, since the check is for equality with the address of the ERC721 token.

Path:
./contracts/ethereum/core/HYPCSwap.sol : addNFT()

Recommendation: Change the revert message to reflect the meaning of the equality check.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I05. Best Practice Violation

In every contract the short form of *uint* is used instead of using the explicit form *uint256*.

Using the explicit format of *uint* makes the code more readable, gas efficient and prevents issues with cross-contract compatibility.

Path:
./contracts/ethereum/core/HYPCSwap.sol
./contracts/ethereum/core/CHYPC.sol
./contracts/ethereum/core/CrowdFundPoolHYPC.sol

Recommendation: Declare all *uint* variables with an explicit type form.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I06. Missed Modifier

In CHYPCNFT.sol, the same check is done multiple times:

```
require(inited == true, "Must be initialized");
```

This code can be moved to a new modifier. This will make the code more readable and clear.

The same occurs in HyperCycleSwap.sol :

require(bytes(HYPCNFT.getAssignment(nftID)).length == 0,

require(reentryLock == false, "No reentry.");

In CrowdFundHYPCPool.sol, the function changeAssignment() uses the following check :

require(msg.sender == proposals[proposalIndex].owner, "Must be owner of proposal.");

instead of using the existing modifier :

modifier proposalOwner(uint proposalIndex)

Paths:
./contracts/ethereum/core/CHYPC.sol : mint(), burn(), assign()
./contracts/ethereum/core/HyperCycleSwap.sol : addNFt(), swap(), redeem()
./contracts/ethereum/core/CrowdFundHYPCPool.sol : changeAssignment()

Recommendation: Create a new modifier to remove code repetition.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I07. Unused Identifier

The field *chypcAmount* was declared in the *ContractProposal* structure but is never used.

Unused identifiers lead to increasing deployment Gas costs, decreased storage optimization, and decreased code quality.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : ContractProposal

Recommendation: Remove unused struct variable.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I08. Constant Variables

The field *requestedAmount* was declared in the *ContractProposal* structure but always stores the same value 524288000000.

This leads to inefficient gas usage.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : ContractProposal

Recommendation: The variable can be declared as constant and removed from the structure.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I09. Redundant Check

The createdDeposit() function contains an allowance check:

```
require(
        HYPCToken.balanceOf(msg.sender) >= amount,
        "Not enough HYPC balance."
       )
```

This is unnecessary because the same checks are already implemented in the transferFrom() function of the ERC20 token.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : createDeposit()

Recommendation: Remove redundant code.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I10. Floating Pragma

The project uses floating pragmas *^0.8.2*.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Paths:
./contracts/ethereum/interfaces/ICHYPC.sol
./contracts/ethereum/interfaces/IHYPC.sol
./contracts/ethereum/interfaces/IHYPCSwap.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I11. Contradiction

The comment in the IHYPC interface specifies : “Accesses the mint and burn functions of the HYPC contract”, but only the burn function is defined.

Path:
./contracts/ethereum/interfaces/IHYPC.sol

Recommendation: Align the documentation and the code implementation.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I12. Unused Import

OpenZeppelin's Ownable is inherited but never used.

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

Recommendation: Remove redundant inheritance to save GAS on deployment and increase the code quality.

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I13. 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.

Paths:
./contracts/ethereum/core/HYPCSwap.sol : addNFT(), swap(), redeem()
./contracts/ethereum/core/CHYPC.sol : initContract(), mint(), burn(), assign(), getAssignment(), getBurnData()
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : createProposal(), createDeposit(), cancelProposal(), withdrawDeposit(), updateDeposit(), finishProposal(), changeAssignment(), getUserDeposits(), getDeposit(), getDepositLength(), getProposal(), getProposalsLenth()

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

Found in: e60234e

Status: Fixed (6e1b7e88638393a2a3877f8e8fa48aef76816f57)

I14. Unused Identifier

The variable *endTime* is declared in the code but is never used.

Unused identifiers lead to increasing deployment Gas costs, decreased storage optimization, and decreased code quality.

Path:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol : completeProposal()

Recommendation: Remove unused variable.

Found in: 6e1b7e8

Status: Fixed (22ecf5413d10e2d5ef3b6153b65ca80d92027671)

I15. Typos

There are various typos in the comments.

CrowdFundHYPCPool:
whena -> when a
speified -> specified
liqudity -> liquidity
transfered -> transferred
acculumated -> accumulated
mutliplies -> multiplies
compeleted -> completed
tokend -> token

HyperCycleSwap:
recieved -> received
itterate -> iterate

Paths:
./contracts/ethereum/core/CrowdFundPoolHYPC.sol
./contracts/ethereum/core/HyperCycleSwap.sol

Recommendation: Fix typos.

Found in: 6e1b7e8

Status: Fixed (22ecf5413d10e2d5ef3b6153b65ca80d92027671)

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.

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.

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.

Appendix 2. Scope

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

Initial review scope

Repository	https://gitlab.com/dliendo05/hypc-polygon/-/tree/develop
Commit	e60234e9e8558e05f4614ed25a9ba883cec565d9
Whitepaper	https://www.hypercycle.ai/_files/ugd/54374c_4641b8d27f8343a190315699371bb042.pdf
Requirements	https://gitlab.com/dliendo05/hypc-polygon/-/blob/develop/description_for_audit.txt
Technical Requirements	https://gitlab.com/dliendo05/hypc-polygon/-/blob/develop/README.md
Contracts	File: ./contracts/ethereum/core/CHYPC.sol SHA3: e65a80bf44a5c5d08b4e640da9a1b6d183e613b5691b14031151cda9d77aaaa9 File: ./contracts/ethereum/core/CrowdFundPoolHYPC.sol SHA3: 2e29758c34914bbc0f4d4af16278c5fbee4ce537d9985868e60e06fd19773042 File: ./contracts/ethereum/core/HYPCSwap.sol SHA3: f642c9a47d46722171116e37400fcb69a7418c561dda17afa95981145b43d158 File: ./contracts/ethereum/interfaces/ICHYPC.sol SHA3: b4796afe4a2ee5bfdc1bd0974264abad32ca8df236c3577cec296febe80ef5cb File: ./contracts/ethereum/interfaces/IHYPC.sol SHA3: 582fe0244acbe2a23a3065e1b54353b9a9c4d5c4dd21db7a4629775fce0deddd File: ./contracts/ethereum/interfaces/IHYPCSwap.sol SHA3: 37048967b9f8fe5b738645460189ff98bbaf5fc1909b6b38fddf006555061217

Second review scope

Repository	https://gitlab.com/dliendo05/hypc-polygon/-/tree/develop
Commit	6e1b7e88638393a2a3877f8e8fa48aef76816f57
Whitepaper	https://www.hypercycle.ai/_files/ugd/54374c_4641b8d27f8343a190315699371bb042.pdf
Requirements	https://gitlab.com/dliendo05/hypc-polygon/-/blob/develop/description_for_audit.txt
Technical Requirements	https://gitlab.com/dliendo05/hypc-polygon/-/blob/develop/README.md
Contracts	File: ethereum/core/CHYPC.sol SHA3: 23bf148025ddbfe7961dd0e3f7ba0eea2106a9491cc55078d0e842cd5e92782f File: ethereum/core/CrowdFundHYPCPool.sol SHA3: d149c38d935436a05e0e81527ca988a361523eac64894c4e1b8660381944bda5

	File: ethereum/core/HyperCycleSwap.sol SHA3: 5e5220aee79b98671b4700a57c9824de8872f7d9528eaee00b925104f33614cd File: ethereum/interfaces/ICHYPC.sol SHA3: 5d99d9039c7a167e42c1893f47b8419e1cb4a37e81c4dc0fc1d0a8c05112a171 File: ethereum/interfaces/IHYPC.sol SHA3: 62669545a79fb56604eebc6c18aefec58b76af32bf21affe1b2077ddfb1f4d9f File: ethereum/interfaces/IHYPCSwap.sol SHA3: 1157deaf935903ef86a40200f52e94743952b5c189128a997102ab8d68a616c2

Third review scope

Repository	https://gitlab.com/dliendo05/hypc-polygon/-/tree/develop
Commit	22ecf5413d10e2d5ef3b6153b65ca80d92027671
Whitepaper	https://www.hypercycle.ai/_files/ugd/54374c_4641b8d27f8343a190315699371bb042.pdf
Requirements	https://gitlab.com/dliendo05/hypc-polygon/-/blob/develop/description_for_audit.txt
Technical Requirements	https://gitlab.com/dliendo05/hypc-polygon/-/blob/develop/README.md
Contracts	File: contracts/ethereum/core/CHYPC.sol SHA3: 85508b2f6cd70ac506338f8a1d68fcebc103532f00ba2c0ba8f5c5c0776ad00b File: contracts/ethereum/core/CrowdFundHYPCPool.sol SHA3: 4e2b886395f805bd9f14ebe54f6451fd8fea9de268505e037540d9a49ab9aa8f File: contracts/ethereum/core/HyperCycleSwap.sol SHA3: b9d09462abd66720a9fae159b654c3396edc0b51cfaf5081f0e10ea6a9b49f8c File: contracts/ethereum/interfaces/ICHYPC.sol SHA3: 5d99d9039c7a167e42c1893f47b8419e1cb4a37e81c4dc0fc1d0a8c05112a171 File: contracts/ethereum/interfaces/IHYPC.sol SHA3: 62669545a79fb56604eebc6c18aefec58b76af32bf21affe1b2077ddfb1f4d9f File: contracts/ethereum/interfaces/IHYPCSwap.sol SHA3: 1157deaf935903ef86a40200f52e94743952b5c189128a997102ab8d68a616c2