PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Preliminary Report

For HyperCycle (Share Tokens)

14 December 2023

paladinsec.co   info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for HyperCycle's Share Tokens contract on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	HyperCycle
URL	https://www.hypercycle.ai/
Platform	Ethereum
Language	Solidity
Preliminary Contracts	https://github.com/hypercycle-development/hypercycle-contracts/blob/aeac0af11ca02f1aa2b94f6ffb4b1f418b626813/contracts/ethereum/core/HyperCycleShareTokens.sol
Resolution	https://github.com/hypercycle-development/hypercycle-contracts/blob/85bef6073b05602b0ed68eb66a8d599eabdbaf16/contracts/ethereum/core/HyperCycleShareTokens.sol

1.2 Contracts Assessed

Name	Contract	Live Code Match
`HyperCycleShareTokens`	`0x9d21A23cD33d7e56a69e0b103828dd5A67f50666`	UNMATCHED

The client implemented additional functionalities that did not fall within the original scope of audit. The client stated that these additional features and changes have been reviewed by another third-party auditor, however, they have not been verified or validated by the Paladin team. Please refer to their report for more details.

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● Governance	0	-	-	-
● High	1	1	-	-
● Medium	1	-	1	-
● Low	4	4	-	-
● Informational	3	3	-	-
Total	**9**	**8**	**1**	-

Classification of Issues

Severity	Description
● **Governance**	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
● **High**	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● **Medium**	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● **Low**	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● **Informational**	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 HyperCycleShareTokens

ID	Severity	Summary	Status
01	HIGH	License holders that do not deposit cHyPC tokens upon shares creation will have their license token locked forever	✓ RESOLVED
02	MEDIUM	Reward distribution is prone to front-running	PARTIAL
03	LOW	Missing input validation of share ids on ERC1155 transfers	✓ RESOLVED
04	LOW	Assigned license number upon share creation is not unassigned when the share is closed	✓ RESOLVED
05	LOW	Checks-Effects-Interactions pattern is not adhered to	✓ RESOLVED
06	LOW	Arithmetic issues	✓ RESOLVED
07	INFO	Constructor input validation check for `endNumber` can be added	✓ RESOLVED
08	INFO	Insufficient input validation for ownership transfer	✓ RESOLVED
09	INFO	Typographical issues	✓ RESOLVED

2 Findings

2.1 HyperCycleShareTokens

`HyperCycleShareTokens` is an ERC1155 contract that allows `HyperCycleLicense` NFT holders to lock their license tokens as well as cHyPC NFT tokens in exchange for contract shares so that they can share revenue with other participants in the HyperCycle system.

Two types of shares are minted with each deposit: revenue shares and wealth shares. While the wealth shares are not used in the current version of the `HyperCycleShareTokens` contract, the revenue shares determine the `HyPC` token portion of the total revenue that holders can claim at any point in time.

Revenue tokens can be transferred between different addresses, and for that reason, the `claimRevenue` operation is executed for both the sender and receiver on each transfer.

After a minimum of 24 hours passes since the creation of a share, the creator or current owner of this share can close it, which will halt deposits of revenue tokens for this particular share number and return the locked license and `cHyPC` NFT tokens back to the current owner.

There are also additional functionalities that allow the creator of a share to change the owner and set a message associated with the share number.

The owner of the `HyperCycleShareTokens` contract can manage (increase) the soft limit of shares that can be created.

Important Resolution Update

After the main audit round was conducted, the client implemented additional functionalities that did not fall within the original scope of audit. The client stated that these additional features and changes have been reviewed by another third-party auditor, however, they have not been verified or validated by the Paladin team.

Paladin has only verified the resolutions for the issues and/or vulnerabilities which were raised during our initial audit, thus we are unable to give any opinion regarding the safety of the newly-added features. Users should refer to the audit report for more information. The out-of-scope features consists of:

- `_check CHYPCAssignment`
- `burnRevenueTokens`
- `burnWealthTokens`
- `getRevenueTokenTotalSupply`
- `getWealthTokenTotalSupply`

2.1.1 Privileged Functions

- `renounceOwnership [onlyOwner]`
- `transferOwnership [onlyOwner]`
- `increaseShareLimit [onlyOwner]`
- `transferOwner [shareOwner]`
- `cancelShareTokens [shareOwner]`
- `setShareMessage [shareOwner]`

2.1.2 Issues & Recommendations

Issue #01	**License holders that do not deposit cHyPC tokens upon shares creation will have their license token locked forever**
Severity	● HIGH SEVERITY

Description	The HyperCycleShareTokens contract is an ERC1155 contract that accepts a deposit of a license NFT and a cHyPC NFT.
	
	As described in the code documentation by the HyperCycle team, while the typical use case is that the original creator of the share deposits both the license NFT and cHyPC tokens into the share contract, there are some use cases where it might make sense to only deposit the license instead. This could be the case where this creator is a smart contract that used the CrowdFundHYPCPoolV2 contract to get an assignment pointed to the license id. Thus, the smart contract would have a guarantee that the license has backing cHyPC but does not own the cHyPC itself. In this case, a share can be created instead with a cHyPC id of 0, which will bypass the cHyPC retrieval and assignment.
	
	The implementation of the above mechanism to allow optional deposits of cHyPC tokens looks as follows in the createShareTokens function:
	
	L336-340
	`licenseContract.safeTransferFrom(to, address(this), licenseNumber);`
	`if (chypcNumber > 0) {`
	`    swapV2Contract.safeTransferFrom(to, address(this), chypcNumber);`
	`    swapV2Contract.assignNumber(chypcNumber, licenseNumber);`
	`}`
	
	If the share creator passes a chypcNumber of 0, no cHyPC token will be locked in the HyperCycleShareTokens contract. However, if we look at the cancelShareTokens function implementation, the above check is missing and the ERC721.safeTransferFrom method is called regardless of whether a cHyPC token has been locked upon creation:

	L373-374
	`swapV2Contract.safeTransferFrom(address(this), msg.sender, chypcNumber);`
	`licenseContract.safeTransferFrom(address(this), msg.sender, licenseNumber);`
	
	Since no cHyPC was deposited during the share creation, the `chypcNumber` will have a value of 0 which is not an ID of a token held by the `HyperCycleShareTokens` contract. Therefore, the call to cancelShareTokens will revert, effectively locking the license NFT indefinitely.

Recommendation	Consider executing `swapV2Contract.safeTransferFrom` within `createShareTokens` only if the `chypcNumber` is not 0.
Resolution	✓ RESOLVED A `chypcTokenHeld` variable is now included within the `shareData` struct of a `shareNumber`, which is set to `true` if the share is created with a `HYPCSwapV2` token. In this scenario, a special callpath is followed whenever the share is canceled, which will then transfer the NFT out if it was in fact set as `true`.

Issue #02	Reward distribution is prone to front-running
Severity	● MEDIUM SEVERITY
Description	Any user can deposit rewards for an active `shareNumber`, which will immediately increase the `revenueDeposited` value. Since this value is directly used to calculate the pro-rata rewards, a malicious user can simply front-run a large revenue deposit by purchasing a majority of the tokens and gather a large amount of the revenue which has been deposited.
Recommendation	The most elegant solution for this issue is to modify the contract such that it supports a time-based reward distribution methodology. However, since such a change will be quite intrusive and basically changes the whole reward calculation and distribution logic, a careful re-audit would be required to guarantee the security of the smart contract. The whole issue depends on the economical incentive behind it — if for example, fees are involved in such a purchase (if it is listed on a corresponding marketplace), this issue could also be resolved by increasing these fees, or simply keeping the deposited revenue per call low and splitting it over several hours per day. **Do note that for significant code changes in the resolution round, a revalidation fee may apply.**
Resolution	● PARTIALLY RESOLVED A reward delay mechanism has been implemented which allows the token owner to set an arbitrary delay time after which the reward will be assigned. However, this does not 100% mitigate the risk since a malicious user can still just front-run a large revenue deposit upon unlocking.

Issue #03	Missing input validation of share ids on ERC1155 transfers
Severity	● LOW SEVERITY

Location	L257-260
	`modifier shareExists(uint256 shareNumber) {`
	`    if (shareNumber < startShareNumber || shareNumber >= currentShareNumber) revert ShareDoesntExist();`
	`    _;`
	`}`

Description	The `HyperCycleShareTokens` contract implements the following modifier to prevent users from calling any methods with share ids that have not been created yet. The modifier is applied on all corresponding functions except for the overridden `safeTransferFrom` and `safeBatchTransferFrom`. This would allow users to transfer shares and claim revenue for invalid share ids, which should not be the expected behavior. However, since such shares are never minted, the only amount users will be able to transfer is 0 which will just emit faulty events.
Recommendation	Consider adding the `shareExists` modifier to the `safeTransferFrom` and `safeBatchTransferFrom`. Additionally, consider checking that the value transferred is always greater than 0.
Resolution	✓ RESOLVED The following check was added in both functions: `if (shareData[shareNumber].status == Status.NOT_CREATED) revert ShareDoesntExist();`

Issue #04	Assigned license number upon share creation is not unassigned when the share is closed
Severity	● LOW SEVERITY

Description	Within `createShareTokens`, `licenseNumber` is assigned to the deposited chypcNumber via the following call:
	
	L339
	`swapV2Contract.assignNumber(chypcNumber, licenseNumber);`
	
	If we look at the implementation of the `assignNumber` method, we see that it first un-assigns the previously assigned `targetNumber` and then assigns the new one if it is greater than 0.
	
	`function assignNumber(uint256 tokenNumber, uint256 targetNumber) external isOwnerOf(tokenNumber) isCalledInThisBlock(tokenNumber) {`
	`    _unassign(tokenNumber);`
	
	`    if (targetNumber > 0) {`
	`        [...]`
	
	However, when the share is closed, chypcNumber and `licenseNumber` are returned back to the creator (or current owner) of the share without un-assigning it in the `swapV2Contract`.

Recommendation	Consider whether the `licenseNumber` should continue to be assigned to the chypcNumber after the share is canceled. If not, add the following line within `cancelShareTokens`: `swapV2Contract.assignNumber(chypcNumber, 0);`
Resolution	✓ RESOLVED The recommendation was implemented.

Issue #05	Checks-Effects-Interactions pattern is not adhered to
Severity	● LOW SEVERITY
Description	Within `createShareTokens`, the two calls to `_mint` are done before the end of the function, which is unsafe due to the `onERC1155Received` hook called on the receiver (the untrusted `msg.sender` in this case) during each mint. However, no re-entrancy vulnerability that could be exploited in the `HyperCycleShareTokens` contract was found during the review, so we rated this issue as low severity.
Recommendation	Consider moving the `_mint` calls after the other "trusted" external calls and before the event emission.
Resolution	✓ RESOLVED The recommendation was implemented.

Issue #06	Arithmetic issues
Severity	● LOW SEVERITY

Description	The revenue portion earned by a given share holder is calculated within the `_claimRevenue` function in the following manner:
	
	L413-415
	`uint256 ownershipRatio = RATIO_DECIMALS*balanceOf(claimerAddress, rTokenNumber)/ REVENUE_TOKEN_MAX_SUPPLY;`
	
	`uint256 amountToGiveAddress = (revenueDeposited-lastShareClaimRevenue[shareNumber][claimerAddress])*ownershipRatio/RATIO_DECIMALS;`
	
	`ownershipRatio` is calculated by using the claimer revenue shares balance, the total supply of revenue tokens and a `RATIO_DECIMALS` constant which has a value of `1e6`.
	
	The result is then used when calculating the amount of `HyPC` tokens the claimer should receive. However, due to division rounding and low precision, the result may be a few `wei` less than the actual amount.
	
	Moreover, it must be considered that this calculation comes with a natural limitation, such that if a user holds less than ~0.0001% of the `totalSupply` (2**20).

Recommendation	While the above formula works in practice as expected, the small disadvantages should be kept in mind. If desired, the client can rewrite this functionality, potentially with a larger scaling factor.
Resolution	✓ RESOLVED The scaling factor `RATIO_DECIMALS` was increased to `10**12`.

Issue #07	Constructor input validation check for `endNumber` can be added
Severity	● INFORMATIONAL

Description	The owner of the contract can increase the soft limit of share numbers that can be minted ranging from the `startShareNumber` to the `endShareNumber`.
	
	`currentShareNumber` is then used in the `createShareTokens` method to determine the ids of the revenue and wealth share tokens minted:
	
	L315-318
	`uint256 shareNumber = currentShareNumber;`
	`uint256 rTokenType = shareNumber*2;`
	`uint256 wTokenType = shareNumber*2+1;`
	`currentShareNumber+=1;`
	
	If the `currentShareNumber` reaches a value of `type(uint256).max / 2 + 1`, the calls to `createShareTokens` will revert due to arithmetic overflow even if the `endShareNumber` is greater than `type(uint256).max / 2 + 1`.

Recommendation	Consider implementing the following check in the `HyperCycleShareTokens` contract constructor: `if (endNumber * 2 + 1 > type(uint256).max) revert InvalidShareNumberRange();`
Resolution	✓ RESOLVED

Issue #08	Insufficient input validation for ownership transfer
Severity	● INFORMATIONAL
Description	`transferOwner` does not implement a zero address check nor the two-step ownership transfer pattern.
Recommendation	Consider checking for the zero address in `transferOwner`.
Resolution	✓ RESOLVED A check was added for `address(0)`.

Issue #09	Typographical issues
Severity	● INFORMATIONAL
Description	It is redundant to use the `SafeERC20` library with known standard `ERC20` token contracts such as `HYPC`. — 24 hours can be used instead of `60*60*24*` for `MIN_SHARE_DURATION` to improve readability. — There are several sections within the contract where a line ends with 1 or multiple spaces. Consider using the following regex in your IDE to identify those lines: `\s+?$`. — The following spelling mistakes were noted: L43: "call is made both" should be "call is made for both". L71: "ther are some…" should be "there are some" L73: "contracto" should be "contract to" and "garauntee" should be "guarantee". L109: "sort" should be "soft". L169: "functon" should be "function". L197: "the" should be "The". L209: "transferred to ." should be "transferred to." L212: "reveue" should be "revenue". L232: "event EarningsWithdrawl" should be event "EarningsWithdrawal". L255: "the give share" should be "the given share". L351, L519: "transfering" should be "transferring". L361: "exisiting" should be "existing". L530: "contact" should be "contract".

Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED Most of the issues were fixed.

PALADIN
BLOCKCHAIN SECURITY