DcentraLab Diligence

dcentralab.com/diligence

Audit Report

HyperCycle

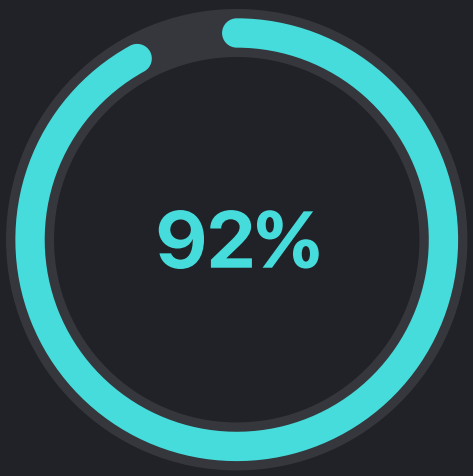
https://www.hypercycle.ai

Provided By DcentraLab Diligence on September 28, 2023

Security Audit Score

Pass

DcentraLab Diligence team has conducted an extensive audit on HyperCycle Contracts and has found the code to be in low risk level given proper deployment and multi-sig permissioning.



- Low Risk
- Small Risk
- Medium Risk
- High Risk

Scope

Audited Repository:

https://github.com/hypercycle-development/hypercycle-contracts

Audited Branch:

license_develop

Audited Commit Hash:

ffdeb9e5b7b659b21aed87c1d6c0dced7991336f

Audited Contracts:

CrowdFundHYPCPoolV2.sol

Contracts Reviewed For Context:

- CHYPC.sol
- HyperCycleSwap.sol
- HyperCycleToken.sol
- CrowdFundHYPCPool.sol

Fixes Commit Hash:

7f0d455fdb632467448b4728504cb463a2534a29

Extended Scope Commit Hash:

3edc1019914acc2e7c902ecb11a7000e48871ce9

Extended Scope Audited Contracts:

HyperCycleLicense.sol

Extended Scope Fixes Commit Hash:

76488fba30a6ffa283d396041aed809240237e9e

General Contracts Outline

Ethereum Contracts:

HyperCycleLicense.sol:

- Purpose: Manages specialized NFTs called "Licenses" originally created on the Cardano chain.
- Features: Allows importing a license from Cardano, setting a Cardano transaction ID, and splitting NFTs into child tokens.

CHYPC.sol:

- Purpose: Containerizes HyPC tokens into a more manageable unit called c_HyPC, which is an ERC721 token.
- Features: Allows conversion of 524,288 HyPC into one c_HyPC and vice versa. Additionally, allows for the assignment of a string (usually a license number) to a c_HyPC.
- Interaction: Interacts with HyperCycleSwap.sol for the swap functionality.

HyperCycleSwap.sol:

- Purpose: Manages the swap mechanism between HyPC (ERC20) and CHYPC (ERC721).
- Features: Provides the swap functionality.
- Interaction: Interacts with CHYPC.sol for swapping and CrowdFundHYPCPoolV2.sol for pooling.

CrowdFundHYPCPoolV2.sol:

- Purpose: Acts as a pool for users who can't afford 524,288 HyPC to create proposals and acquire CHYPC tokens.
- Features: Allows for proposals with a lesser amount of HyPC as collateral. This amount acts as interest for others who contribute to the proposal.
- Interaction: Directly depends on CHYPC.sol and HyperCycleSwap.sol for its functionalities.

HyperCycleToken.sol:

- Purpose: Appears to be the ERC20 token (HyPC) contract.
- Interaction: Interacts with CHYPC.sol and HyperCycleSwap.sol for token conversions and swapping.

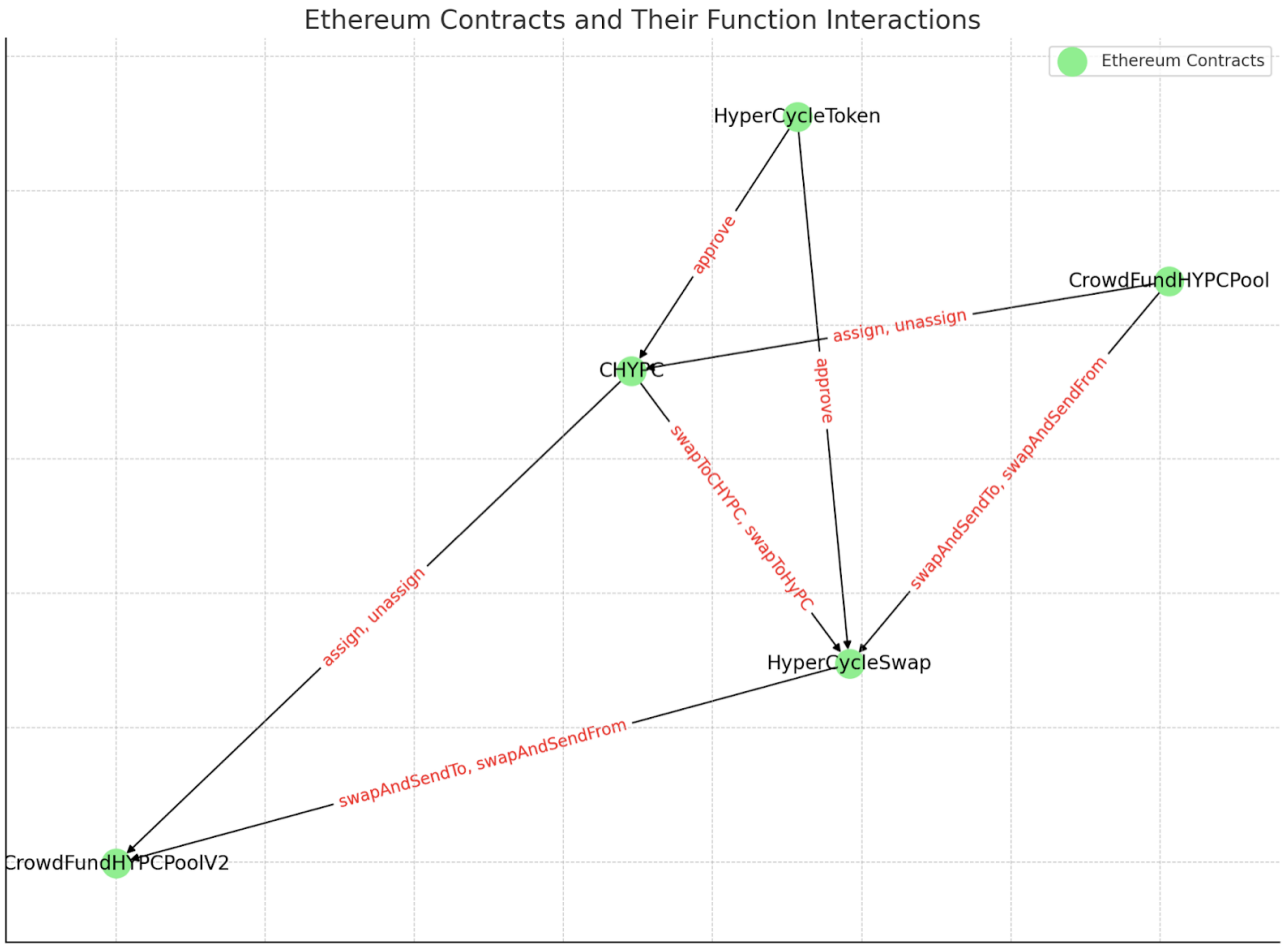
CrowdFundHYPCPool.sol:

- Purpose: The predecessor to CrowdFundHYPCPoolV2.sol, it provides similar functionalities but with limitations like not supporting multiple NFTs in a single proposal.
- Interaction: Likely interacts with CHYPC.sol and HyperCycleSwap.sol similarly to its V2 counterpart.

Interactions:

Ethereum Contracts:

- CHYPC.sol interacts with HyperCycleSwap.sol to manage the conversion between HyPC and CHYPC.
- CrowdFundHYPCPoolV2.sol depends on CHYPC.sol and HyperCycleSwap.sol to facilitate the pooling and conversion of tokens.



General Notices:

1. Maintaining the license id taxonomy uniqueness and validity is critical to the sanity and coherence of the system. However, there is no strong enforcement mechanism for verifying license IDs are uniquely managed between their cardano and ethereum instances.
2. Deployment: should include script to hand over ownership to multi-sig of cold-storage vault/ congress contract
3. Deployment: should include checksum scripts for making sure ownership was correctly transferred, and correct contracts set

Risks:

DcentraLab Diligence (DD) has performed all checks and verifications in its capacity to ascertain the safety of the code. However, it should be noted that misuse of the code, bad deployment practices, bad key management, exposing of private keys of the deployer and/or owner address and/or multi-sig signer addresses and/or fee collector address and/or any exposition of the code to malicious actors may result in an exploit of the code and loss of state and/or funds. Furthermore, there is always a chance that other Smart Contracts code could be written and deployed to cause the provided code by DD to act outside the intended scope by the client, to the point of causing state corruption or loss of funds to the client of the users of the code.

Issues Severity Reference Table

Type

Discussion

The issue severity is dependent on design, centralization, and product specifications of the project.

Informational

This issue is not critical and does not pose an immediate threat to the functionality or security of the smart contract. It is simply an informational item that the auditors have identified and recommends addressing for best practices or to improve the overall performance of the contract.

Low

This issue is relatively minor and does not pose a significant risk to the functionality or security of the smart contract. While it is recommended to address these issues to ensure the highest level of quality and security, they are not likely to cause significant problems if left unaddressed.

Medium

This issue poses a moderate risk to the functionality or security of the smart contract. While it may not be immediately exploitable, it has the potential to cause problems in the future if left unaddressed. It is recommended to address these issues as soon as possible to prevent any potential negative impact on the contract.

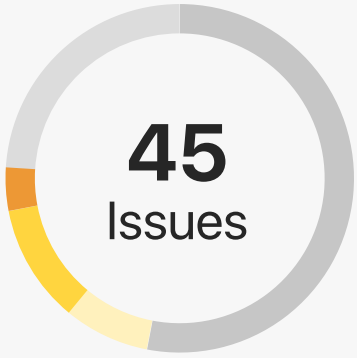
High

This issue poses a significant risk to the functionality or security of the smart contract. Addressing these issues as soon as possible is recommended to prevent any potential negative impact on the contract. Failure to address these issues could result in significant problems and potential loss of funds or other assets.

Critical

This issue poses an immediate and severe risk to the functionality or security of the smart contract. It is recommended to address these issues immediately to prevent any potential negative impact on the contract. Failure to address these issues could result in catastrophic problems and significant loss of funds or other assets.

Findings Summary



- Discussion
- Informational
- Low Risk
- Medium Risk
- High Risk
- Critical Risk

ID	Title	Severity	Status
E.1	Dependency conflicts	Informational	Resolved
E.2	Dependency vulnerabilities	Informational	Resolved
E.3	.env.example file lacks an attribute	Informational	Resolved
E.4	Code formatting	Informational	Resolved
G.1	Use of 'require' statements with solidity compiler version ^0.8.4	Informational	Resolved
G.2	Non differentiated token precision and calculation precision	Low	Resolved
G.3	License management	High	Partially Resolved
G.4	Contract non-upgradeability and immutability	Discussion	Acknowledged
A.1	Additional optimization of 'amount' local variable usage	Informational	Resolved
A.2	Proposal state constants can be made an enum	Informational	Resolved

Findings Summary

ID	Title	Severity	Status
A.3	Constructor missing event	Informational	Resolved
A.4	Proposal deadline argument can be used to create an unfulfillable proposal	Low	Resolved
A.5	Missing minimum APR threshold which allows for 0% returns	Discussion	Acknowledged
A.6	vague reference to amount parameters' token fragmentation state	Informational	Acknowledged
A.7	Missing upper border check on proposal creation on 'numberNFTs' argument	Low	Resolved
A.8	Lack of flexibility and security impact of non-upgradeability and immutable values	Discussion	Acknowledged
A.9	Lack of flexibility with 3 offered 'termLength' options	Discussion	Acknowledged
A.10	Missing check for existing proposals of the message sender	Low	Acknowledged
A.11	transfer()/transferFrom() optimization	Informational	Resolved
A.12	Possible errors once deposit reaches the requested amount	Medium	Resolved
A.13	Unnecessary double reference on transferDeposit()	Informational	Resolved

Findings Summary

ID	Title	Severity	Status
A.14	Rules of proposal transfer	Discussion	Acknowledged
A.15	Proposal starting permissions	Low	Resolved
A.16	Computation optimization	Informational	Resolved
A.17	Computation optimization	Informational	Resolved
A.18	Possible proposal and deposit transfers to non-accessible wallets	Medium	Resolved
A.19	volatile fragmented/duplicated code	Medium	Acknowledged
A.20	lack of validations on data assigned to hypc nft	Discussion	Acknowledged
A.21	Unused library import	Informational	Resolved
A.11	transfer()/transferFrom() optimization	Informational	Resolved
B.1	Redundant nonReentrant modifier	Informational	Resolved
B.2	Unnecessary use of '_safeMint()'	Informational	Resolved
B.3	Inconsistent updates of 'totalTokens'	Medium	Resolved
B.4	Redundant value update	Informational	Resolved
B.5	Improper event argument names	Informational	Resolved

Findings Summary

ID	Title	Severity	Status
B.6	Redundant value assignment	Informational	Resolved
B.7	Redundant license status check	Informational	Resolved
B.8	Ownership should be assigned to a secure multi signature wallet	Discussion	Resolved
B.9	Missing 'constant' keyword	Informational	Resolved
B.10	'mint' function return value	Discussion	Resolved
B.11	Unoptimized computation	Informational	Resolved
B.12	Burning of child licenses can cause permanent nonexistence of parent licenses	Medium	Acknowledged
B.13	Additional burn event	Informational	Resolved
B.14	Hardcoded value	Informational	Resolved
B.15	Binary tree and license numbers	Discussion	Resolved
B.16	Merge Dynamics and Use Cases	Discussion	Resolved

Complete Analysis

Environmental Findings:

ID E.1: **Status: Resolved**

Informational | Dependency conflicts

Present at: package.json & package-lock.json

Description: Dependency conflicts found in repository forcing npm installation with –force flag.

Recommendation: Resolve dependency conflicts and make sure that none are deprecated or redundant.

ID E.2: **Status: Resolved**

Informational | Dependency vulnerabilities

Present at: package.json & package-lock.json

Description: During the project initialization process warnings showed that vulnerable and deprecated dependencies persist in the repository.

Recommendation: Make sure to update all vulnerable and deprecated dependencies.

Additional Notice: Issue is marked with 'informational' flag as dependencies which are vulnerable are not in direct relation with smart contracts.

Complete Analysis

Environmental Findings:

ID E.3: **Status: Resolved**

Informational | .env.example file lacks an attribute

Present at: .env.example

Description: In the .env.example file there is missing attribute 'ETHEREUM_RPC_PROVIDER' which is required in hardhat.config.ts.

Recommendation: Make sure to add a missing attribute to the file in order to avoid confusion in the process of repository setup.

ID E.4: **Status: Resolved**

Informational | Code formatting

Present at: Throughout the codebase

Description: Even though code present in the repository is of great readability and overall good quality, it could use a standardized formatting in order to straighten out minor imperfections that are present.

Recommendation: Consider formatting the code using standardized tools such as a part of your development process.

Project comment: added prettier to this commit: 54f9f4fa6a8544a377ffc67ac3d47180845e0a7a

Complete Analysis

General Contract Findings:

ID G.1: **Status: Resolved**

Informational | Use of 'require' statements with solidity compiler version ^0.8.4

Present at: Contracts directory

Description: Your chosen compiler version is above 0.8.4 (version that introduced 'custom errors'), this implies that you can reduce gas consumption and bytecode size by introducing them instead of 'require' statements. After introduction of 'custom errors', the 'require' statements are considered redundant as there is no additional value that they provide.

Helpful Resource: https://soliditylang.org/blog/2021/04/21/custom-errors/

Recommendation: Replace 'require' statements with 'custom errors'.

ID G.2: **Status: Resolved**

Low | Non differentiated token precision and calculation precision

Description: We've noticed the 6 decimal param may be interchangeably used for both aligning amounts in the hypc token calculations to the fragmented notation and for utilizing a precision framework for general params (e.g. the apr). This can lead to costly errors later due to the ambiguity of usage in this param

Recommendation: Utilize a different param with different decimals for general computation precision requirements.

Complete Analysis

ID G.3: **Status:** Partially Resolved

High | License management

Description: Throughout the architecture it is important that licenses are managed properly cross chain. That implies the following:

- Only the true owner of a license on cardano shall be able to mint an NFT with that license on other chains.
- License numbers shall not be able to repeat among different NFTs
- License numbers shall be legitimate and respect a specific format

Architectural Recommendation: Make sure that license owner on the Cardano side provides a signature which provides proof of ownership of the license as well as his EVM identification address. Once the back-end of yours verifies the signature, it shall provide another to the user. With signature from your back-end, the user is able to mint an nft with his EVM address and with a specific license number. Except for the signature validation, the license is checked for repetition. If license is not repeated and signature is valid, only then an NFT which backs up such license can be minted.Notice: We are well aware of the fact that realization of NFT backup of specific license is an expensive action and therefore only the real license owner has the interest in proposal creation for it. Though, our architectural recommendation prevents identity theft in more extreme cases.

Notice on Contracts: Consider license assigning be a part of a process of proposal creation in order to save gas and make operation clearer to all users by letting them know which license number(s) are they supporting.

Project Response: Skipped for now - will be relevant after license contract audit.
Project Response #2: We decided to do the burning and issuing of the tokens ourselves to make everything easier, so users would have to ask us to exchange their Cardano tokens for the new Ethereum ones.

Resolvement: Project's approach on licenses was changed during the audit therefore this issue is no longer relevant. New Approach should be carefully audited to make sure that it is secure, but is outside of the current scope.

Complete Analysis

ID G.4: **Status:** Acknowledged

Discussion | Contract non-upgradeability and immutability

Description: While in certain cases non-upgradeability can provide additional security, in many cases it is faced as a disadvantage which prevents developers from fixing a production issue.

Recommendation: Think about upgradeability and decide if current architecture suits your needs.

Suggestion: if for example such issues/bugs had been discovered after deployment, or for product advancement you would need to add new functionality, there would be a very messy process to update the token and switch all holders to a new contract, or update to new contracts and migrate state etc.. which is why we advise on separation of logic and state, aka the proxy pattern for upgradability of logic, without hampering with the state. This kind of architecture design is critical for the ability to resolve bugs post prod deployment. The centralization risk that arises can be solved to the degree of your preference by allocating a more or less distributed multi-sig congress/vault for approving upgrades. It can also be community level DAO that approves this etc..

Project Response: Will remain non-upgradable per project decision.

Complete Analysis

Local Contract Findings:
Contract: CrowdFundHYPCPoolV2.Sol

ID A.1: **Status: Resolved**

Informational | Additional optimization of 'amount' local variable usage

Present at: CrowdFundHYPCPoolV2.sol / withdrawDeposit() @ L635-636

Description: Local variable 'amount' which has an assigned value of 'depositData.amount' is used multiple times inside 'withdrawDeposit()'. We believe that variable was introduced with flow optimization as a purpose. Before the introduction of the mentioned variable, 'depositData.amount' is used one more time, which provides an additional complexity to the function logic.

Recommendation: Initialize 'amount' before the first functional usage in order to further optimize the function flow.

ID A.2: **Status: Resolved**

Informational | Proposal state constants can be made an enum

Present at: CrowdFundHYPCPoolV2.sol @ L93-96

Description: Among the constants present in this contract there are 4 representing the state of the proposal. These 4 constants, as they contain only numbers 0 to 3 orderly, can be replaced with an enum. For such a case enum shall be considered a conventional solution, though the current way of functioning is perfectly fine too.

Recommendation: Replace mentioned constants with an enum.

Complete Analysis

ID A.3: **Status: Resolved**

Informational | **Constructor missing event**

Present at: CrowdFundHYPCPoolV2.sol / constructor() @L280-296

Description: Logic of the 'setPoolFee()' function emits an event once 'poolFee' is set. You might want to emit the event in the constructor, as the value is being set there too. Without an event there you will skip the initial value of poolFee in your event log.

Recommendation: Consider adding 'PoolFeeSet()' event emission after setting 'poolFee' in your constructor logic.

ID A.4: **Status: Resolved**

Low | **Proposal deadline argument can be used to create an unfulfillable proposal**

Present at: CrowdFundHYPCPoolV2.sol / createProposal() @L333

Description: Deadline is required to be greater than block.timestamp which still allows proposal to be unfulfillable.

Recommendation: Consider implementing a time buffer which will not allow for unfulfillable proposals inside the current check.

Solution Implementation Example:
uint256 public constant PROPOSAL_CREATION_DEADLINE_BUFFER = 3600; //One hour
require(block.timestamp + PROPOSAL_CREATION_DEADLINE_BUFFER < deadline);

Complete Analysis

ID A.5: **Status:** Acknowledged

Discussion | Missing minimum APR threshold which allows for 0% returns

Present at: CrowdFundHYPCPoolV2.sol / createProposal()

Description: Backing funds are required to be greater than zero, which still makes an ineffective check as the APR can still be zero.

Recommendation: Set a threshold minimum for the APR and replace current backing funds zero value check with an APR crossing threshold check.

Post fix comment: Removing the check did not fix the mentioned issue.

Project Response: The APR being zero is an acceptable use case of the pool contract and can act as a locking mechanism for HyPC, with the respective parties (depositors and proposer) agreeing to a lock agreement. Alternatively, the proposer could offer off-chain rewards for the depositors. As long as a 0% APR rate doesn't break the contract, this is acceptable. An extra test was added that creates a 0% APR proposal. See 07b90e2a85f6dbffe590eb6fa1222f9fe36c3200

ID A.6: **Status:** Acknowledged

Informational | vague reference to amount parameters' token fragmentation state

Present at: CrowdFundHYPCPoolV2.sol (CrowdFundHYPCPool.sol) & HyperCycleSwap.sol

Description: When it comes to token handling, we recommend to verify that all parameters pertaining to token amounts are in their atomic fragmented notation, and any such parameter or variable that is not, should be clearly notated as being in double/float format. Also note that frontends calculating fragmented notation from user input which is usually in double notation, often result in rounding or casting errors that propagate to the contract and may cause underflow or overflow in intended amounts.

Complete Analysis

Additional Comment: Providing a raw value input (uint with all of its "decimals") to contracts is the most conventional approach. Usually, when it comes to popular dapps, a piece of logic to convert precise floating point numbers to uint is contained on the front-end side. After conversion the front-end will make sure that your wallet receives the exact uint value which represents the float value that you inserted at first. Contract logic should be resilient to frontend rounding/casting errors as much as possible, e.g. by implementing fuzzy limits on logic and calculations

Recommendation: Verify all decimals management is handled off chain via frontend/backend integrations. Expect users to input double notation values, and allow values inputted up to the full fragment precision (e.g. 6 decimals for HYPC), which should then be properly converted by the frontend to uint and prepared for contract interaction.

ID A.7: **Status: Resolved**

Low | Missing minimum APR threshold which allows for 0% returns

Present at: CrowdFundHYPCPoolV2.sol / createProposal() @L335

Description: Argument numberNFTs has no upper border which means someone can request a larger amount than possible to mint.

Recommendation: Consider implementing an upper border check for this argument with a reasonable number.Additional Notice: Make sure that number of NFTs is operable inside of every function that it will go through, complexity of logic will in some places linearly extend based on the number of NFTs in the proposal which may cause crossing block gas limit.

Complete Analysis

ID A.8: **Status:** Acknowledged

Discussion | Lack of flexibility and security impact of non-upgradeability and immutable values

Present at: CrowdFundHYPCPoolV2.sol @L68-74

Description: Values declared in described lines are not changeable, we consider this an intended behavior but wanted to let you know of the potential risks it carries.

Recommendation: Think about potential risks and decide if the current way of functioning is desired.

Project Response: Lack of upgradeability of our contracts is a deliberate choice. Upgradability is done through migrating to new contracts, like from PoolV1 to PoolV2 for example.

ID A.9: **Status:** Acknowledged

Discussion | Lack of flexibility with 3 offered 'termLength' options

Present at: CrowdFundHYPCPoolV2.sol

Description: Though we consider offering only 3 options an intended behavior we wanted to let you know of the lack of flexibility this carries.

Recommendation: Think about how lack of flexibility can impact the application flow and UX during a period of time. Reconsider a behavior which includes dynamic flow and custom time periods.

Project Response: Additional term lengths, if required, will be handled with a future pool versions. At the current time, 36 months is the most reasonable maximum amount of time for a proposal, and 18 months the most reasonable minimum amount of time.

Complete Analysis

ID A.10: **Status:** Acknowledged

Low | Missing check for existing proposals of the message sender

Present at: CrowdFundHYPCPoolV2.sol / createProposal()

Description: The way that current flow works, a single person is able to create a very large amount of proposals.

Recommendation: Consider limiting the amount of active proposals per user to a reasonable number.

Project Response: Limiting per user will not stop a malicious user from creating many proposals since they can just change the address they're using to create proposals. If a user wants to spam the contract with many proposals, then the frontend will just order the proposals by APR and status of the proposals.

ID A.11: **Status:** Resolved

Informational | transfer()/transferFrom() optimization

Present at: CrowdFundHYPCPoolV2.sol

Description: Since the tokens interacting with the architecture belong to you and their logic is known, you can optimize the flow by using regular IERC20 transfer functions instead of safeTransfer. While safeTransfer is a must when handling tokens with unknown logic, when interacting with a token of your own there is no particular need to use this library.

Recommendation: Consider making transfers in raw form in order to optimize gas consumption and reduce bytecode size.

Complete Analysis

ID A.12: **Status: Resolved**

Medium | Possible errors once deposit reaches the requested amount

Present at: CrowdFundHYPCPoolV2.sol / createDeposit() & updateDeposit()

Description: When depositing, the user whose amount is supposed to fill the proposal is prone to reverting as equalization to the requested amount is required. When working with token fragments, amounts on the front-end can easily lose precision and that can cause the mentioned issue.

Recommendation: Consider implementing flow in such a way to accept greater amounts than the proposal owner requested but return the change back to the user if the requested amount is overfilled.

Project Response: This code allows for the final deposit to be less precise to fill the proposal in case of front-end precision issues, but prevents sniping issues where a user wants to deposit the last say, 500,000 HyPC, but then gets sniped by another user that posts 200,000 HyPC. The user posting the 500,000 HyPC didn't ask to make a 300,000 HyPC deposit - maybe they want to only have one deposit to update every two weeks instead of two and would choose to deposit to a different proposal if this was the case.

ID A.13: **Status: Resolved**

Informational | Unnecessary double reference on transferDeposit()

Present at: CrowdFundHYPCPoolV2.sol / transferDeposit() @ L438

Description: Unnecessary reference to a value is made on L438 while that value is stored locally already at L435.

Recommendation: Consider using an already instantiated local variable instead of making a new reference to a value in order to optimize gas consumption and reduce bytecode size.

Complete Analysis

ID A.14: **Status:** Acknowledged

Discussion | Rules of proposal transfer

Present at: CrowdFundHYPCPoolV2.sol / transferProposal()

Recommendation: Think about the rules that should potentially be applied to the flow of proposal transfer, as proposal has different states - those states may imply different behavior in situations.

Project Response: transferring a proposal from one address to another only changes the proposal's owner attribute, which determines what address can perform actions on this proposal. Proposal ownership only impacts transferProposal, cancelProposal, finishProposal, and changeAssignment, with changes to proposal owners not impacting their flow.

ID A.15: **Status:** Acknowledged

Discussion | Proposal starting permissions

Present at: CrowdFundHYPCPoolV2.sol / startProposal()

Recommendation: The time of starting the proposal may carry significance in state or other product dynamics, re-consider to only allow the proposal owner to start their proposal, or to allow the proposal owner to signal once creating the proposal that this proposal is open for anyone to start once fulfilled required amount.

Project Response: Restricting the starting of a proposal to only the proposer gives them the power to stonewall a proposal through either malice, incompetence, or simple forgetfulness. There is a choice between the proposer and the depositor in terms of who to prioritize between interest payments starting "on time" vs the assignment strings being assigned as soon as the proposal starts. Given the nature of the proposer, it is safer to assume they will take on the responsibility to see when the proposals are filled and change their assignment strings when the time comes.

Complete Analysis

ID A.16: **Status: Resolved**

Informational | Computation optimization

Present at: CrowdFundHYPCPoolV2.sol / createProposal() & swapTokens() & updateDeposit()

Description: Inside of mentioned functions there have been unnecessary re-computations of known values on every transaction. Constants PERIODS_PER_YEAR and HYPC_PER_CHYPC are multiplied by SIX_DECIMALS each time they're used in computations.

Recommendation: In order to reduce gas consumption in execution of mentioned functions we recommend to replace redundant computations with constant values.

ID A.17: **Status: Resolved**

Informational | Computation optimization

Present at: CrowdFundHYPCPoolV2.sol / createDeposit() @ L515-516

Description: At mentioned lines the following computation is repeated: 'HYPC_PER_CHYPC_SIX_DECIMALS * proposalData.numberNFTs'

Recommendation: Consider executing computation only once and storing it in a local variable for reuse.

ID A.18: **Status: Resolved**

Medium | Possible proposal and deposit transfers to non-accessible wallets

Present at: CrowdFundHYPCPoolV2.sol / transferDeposit() & transferProposal()

Description: With the occurrence of a simple mistake, the user may transfer his deposit and/or proposal to a wrong wallet.

Complete Analysis

Recommendation: Consider making a 2-step transfer flow, which requires the 'to' wallet to accept the proposal or deposit. That way, in the case of a mistake, the user can just revoke his transfer offer which he made to another wallet.

Project Response: There's a transferRegistry added to prevent fat-fingering.

Auditor's comment: Newly implemented transferRegistry flow provides a similar level of security to our recommended solution.

ID A.19: **Status:** Acknowledged

Medium | volatile fragmented/duplicated code

Present at: CrowdFundHYPCPoolV2.sol / swapTokens()

Description: tokenId retrieved using duplicated segregated logic from swap contract assumed to be aligned with the swap contract, combined with soft assumptions on blocked race conditions for swapped out tokenIds, with no checksum on actual tokenId swapped out. This might also not align in some cases with actual logic executed on swap contract

Recommendation: on swap contract, hold state array for storing tokens ids swapped out in chronological order, expose a getter for nextAvailableTokenId, and add it as input param into swap function, so that on poolv2 you can get the next available token (without code duplication), and send it as param to the swap function, which would checksum and verify this is indeed the next available tokenId to swap out per current state on swap contract. This will ensure a strong handshake on tokenId swapped out between poolv2 and swap contract. Same should be handled for any direct flow of user directly swapping not via the poolv2.

Complete Analysis

Project Response: The CHYPC and Swap contracts are already deployed, and were previously audited by another firm. Since the external contract interfaces are well-defined (developed in house) and static (deployed and not upgradeable), there isn't a possibility for the interface to change. Operations like swapping for a new token can be abstracted into an interface to reduce code duplication, but there is only one place in the contract where this swapping occurs, so we feel there's no need to create an abstraction layer at this time. Likewise, a stronger handshake could be added to use the ERC721Received mechanism to guarantee the tokenId received is correct, but this would be a redundant step in this case. This recommendation will be kept in mind for future versions of the Swap contract.

ID A.20: **Status:** Acknowledged

Discussion | **lack of validations on data assigned to hypc nft**

Present at: CrowdFundHYPCPoolV2.sol / swapTokens()

Description: tokenId retrieved using duplicated segregated logic from swap contract assumed to be aligned with the swap contract, combined with soft assumptions on blocked race conditions for swapped out tokenIds, with no checksum on actual tokenId swapped out. This might also not align in some cases with actual logic executed on swap contract

Recommendation: on swap contract, hold state array for storing tokens ids swapped out in chronological order, expose a getter for nextAvailableTokenId, and add it as input param into swap function, so that on poolv2 you can get the next available token (without code duplication), and send it as param to the swap function, which would checksum and verify this is indeed the next available tokenId to swap out per current state on swap contract. This will ensure a strong handshake on tokenId swapped out between poolv2 and swap contract. Same should be handled for any direct flow of user directly swapping not via the poolv2.

Complete Analysis

ID A.21: **Status: Resolved**

Informational | **Unused library import**

Present at: CrowdFundHYPCPoolV2.sol

Description: Library ‘Strings.sol’ by OpenZeppelin is imported but never used.

Recommendation: Consider removing the library import.

Complete Analysis

Local Contract Findings:
Contract: HyperCycleLicense.Sol

ID B.1: **Status: Resolved**

Informational | Redundant nonReentrant modifier

Present at: HyperCycleLicense.sol / mint() @ L151

Description: At the mentioned line a redundant 'nonReentrant' modifier is present. Its redundancy comes from the fact that the 'onlyOwner' modifier is also applied. Owner is the only wallet suitable for making a reentrancy attack (if it is a contract).

Recommendation: Consider removing the modifier.

ID B.2: **Status: Resolved**

Informational | Unnecessary use of '_safeMint()'

Present at: HyperCycleLicense.sol / mint() @ L158

Description: Usage of mentioned function instead of ordinary '_mint()' is not needed as tokens are being minted to the owner. If the owner is not a contract which is supposed to perform specific action on 'onERC721Received()' call, then safe mint can be replaced with ordinary mint.

Recommendation: Consider replacing '_safeMint()' with '_mint()'.

Complete Analysis

ID B.3: **Status: Resolved**

Medium | Inconsistent updates of 'totalTokens'

Present at: HyperCycleLicense.sol / merge() & split()

Description: Variable 'totalTokens' is being updated on split, number of total tokens is increased by 2. Since the parent token gets burned it would make more sense to increase the number of total tokens by 1. On merge 'totalTokens' is not updated at all while it would make most sense to reduce it by 1, as children tokens are burned and parent token is minted.

Recommendation: If 'totalTokens' variable was meant to track number of existing tokens in circulation consider adapting flow to previously seen description.

ID B.4: **Status: Resolved**

Informational | Redundant value update

Present at: HyperCycleLicense.sol / split() @ L179 & merge() @ L209-210

Description: Attribute 'burnData' is being assigned value of an empty string. In order for the license to get splitted/merged it needs to be minted first, in the 'mint' function, value of an empty string is also set to the mentioned attribute. There might not be a scenario where you need to set this value again in the split/merge functions. 'burnData' value can be set to something other than an empty string only in independent 'burns', which is an action outside of merge/split flow.

Recommendation: Consider removing assignments of an empty string to the 'burnData' attribute in merge and split functions.

Complete Analysis

ID B.5: **Status: Resolved**

Informational | Improper event argument names

Present at: HyperCycleLicense.sol / L118-119

Description: 'Merge' event has the same arguments as 'Split' event and the names of 'child licenses' are not suitable in case of 'Merge'. They're called 'newLicenseId1' and 'newLicenseId2', since these licenses are existing and getting burned, we find this naming improper.

Recommendation: Consider renaming event arguments.

ID B.6: **Status: Resolved**

Informational | Redundant value assignment

Present at: HyperCycleLicense.sol / L84 & L146

Description: Value of 'totalTokens' is being set to zero twice, once in the place of declaration and once in the constructor. Both of these sets are unnecessary as the default value for a newly declared variable in solidity is zero. We understand that leaving the first statement like this can help code be more transparent, but there is no further benefit in setting it twice.

Recommendation: Consider removing the redundant assignment(s).

ID B.7: **Status: Resolved**

Informational | Redundant license status check

Present at: HyperCycleLicense.sol / getBurnData()

Complete Analysis

Description: Function 'getBurnData()' has 2 status checks, one which comes from the 'isValid' modifier and the other one checking that the license is burned. Second status check makes the first one unnecessary, as token being of status 'BURNED' implies that it is not of status 'NOT_MINTED' which is being checked inside the 'isValid' modifier.

Recommendation: Consider removing the modifier.

ID B.8: **Status: Resolved**

Discussion | Ownership should be assigned to a secure multi signature wallet

Description: Have owner as multisig of cold wallets and ensure in deployment script that ownership is transferred to this multi sig immediately post deploy + enable a post deployment structure check script to ensure ownership is indeed held by the multisig.

Project Response: Added this to the deployment script for mainnet.

ID B.9: **Status: Resolved**

Informational | Missing 'constant' keyword

Present at: HyperCycleLicense.sol @ L85-86

Description: 'endRootToken' and 'startRootToken' can be marked as constants.

Recommendation: Declare mentioned variables as constants

Complete Analysis

ID B.10: **Status: Resolved**

Discussion | **'mint' function return value**

Present at: HyperCycleLicense.sol / mint() @ L163

Description: If return value is indeed needed, returning the number of tokens minted does not help to ascertain in verified manner which tokens were actually minted.

Recommendation: first consider if return value actually needed, if needed, consider outputting more precise information such as id range (min,max) of minted license ids.

Project Response: Not needed, so removed.

ID B.11: **Status: Resolved**

Informational | **Unoptimized computation**

Present at: HyperCycleLicense.sol / split() @ L174-175

Description: 'licenseId2' is computed as 'licenseId * 2 + 1' while 'licenseId1' is computed as 'licenseId * 2', this implies that 'licenseId2' can be computed more easily via 'licenseId1 + 1'.

Recommendation: Consider implementing described flow in order to reduce gas usage.

Complete Analysis

ID B.12: **Status:** Acknowledged

Medium | Burning of child licenses can cause permanent nonexistence of parent licenses

Description: Since child licenses can get burned independently outside of merge/split flow, that can cause merge to be impossible later on, as merge requires both children to be non burnt.

Recommendation: If this is not desired behavior consider limiting independent burns only to the parent/root nodes or enable mints of child nodes individually, or enable merges where one of the licenses being merged is burnt.

Project Response: The merge functionality is an optional aspect of the contract and is a "nice to have" feature. If a token holder burns a child token, then it is expected that the parent can no longer be merged in this contract. If, for example, the burn mechanism was used for a one-way bridge to another blockchain, then both tokens would have to be burnt into the new blockchain in order to be merged up there, if merging was supported there.

Conclusion: This is a desired behavior.

ID B.13: **Status:** Resolved

Informational | Additional burn event

Present at: HyperCycleLicense.sol / burn()

Description: Even though native '_burn()' function which is called inside the 'burn()' emits a 'Transfer' event. You might want to add your own specific event which can contain extra arguments.

Recommendation: Consider adding an event.

Complete Analysis

ID B.14: **Status: Resolved**

Informational | Hardcoded value

Present at: HyperCycleLicense.sol / merge() @L202

Description: At the mentioned line there is a hardcoded value which can be replaced by global constant 'startRootToken'.

Recommendation: Consider implementing described changes in order to increase code quality and readability.

ID B.15: **Status: Resolved**

Discussion | Binary tree and license numbers

Description:

1. The notice comment mentions 64 levels supported by the tree, yet the contract only supports 10 levels from height 19 to 10. I.e. There is no explicit adherence for 64 leaves on the contract
2. In documentation it states: "These licenses have a unique LicenseID number, ranging from 8796629893120 to 8796629897215 (4096 in total). But checking 8796629893120+4096 - 8796629897215 the result is 1 so seems there is inconsistency there (perhaps add "exclusive" on the last id in the comment e.g. "from 8796629893120 inclusive to 8796629897215 exclusive)
3. Please review: https://colab.research.google.com/drive/1Vbq2hCqm6rILzisFX2of6T_im4Bdtfvb?usp=sharing - according to this simulation:
   a. each root license can generate up to 1022 sublicenses,
   b. and the total splitting of licenses can produce, including roots, 4,191,231 licenses. The min license id is =8796629893120 (the min root), and the max is 4503874507375103 (bottom right of the tree)
   c. The last level of the tree enabled by this contract is from license id minimum of 4503874505277951 to maximum of 4503874507375103 -> the last level min and max values also differ from what's written in the notice - Tokens at the bottom level in this contract are in the range of 4503874505277440 to 4503874505277951 inclusive

Complete Analysis

Recommendation: Review discrepancies and ascertain which is the intended. If values in notice are incorrect, please remedy.

Project Response: There are 4096 licenses in total, so the last token should be the starting token plus 4096, minus 1 (eg: starting at 10, and creating two tokens will give 10+2-1 = 11, so 10 and 11 would be the tokenIds).
The endRootToken variable should be called endRootTokenLimit instead, since it is the startId plus 4096 and not the actual last root token. I've changed this in the code.
Each split level increases the number of tokens by a factor of 2, so this should be a power of two. Since there are 9 splits, this is 2^9 = 512.
From 2.) The max comes from the 8796629897215 token, which gives 4503874507374591 as the maximum token (a few digits were incorrect previously and are fixed now in the comment).
The minimum tokenId at the bottom level is 4503874505277440 as stated (8796629893120 * 2^9).

Resolution: notes have been modified and all information checksummed.

ID B.16: **Status: Resolved**

Discussion | Merge Dynamics and Use Cases

Description: If the intended act of splitting a license is to distribute it among more people than the original owner, it's fair to assume that on merge, not all licenses are owned by the same addresses. In such case, it could prove useful to enable with approval mechanisms the ability of owner X to merge licenses held by addresses Y and Z (after they provided approval for X to transfer their licenses), so that it would be possible to merge 2 licenses held by different people into a single license held by either one of them or a third address.

Recommendation: As the contract is non upgradable please think of any potential future usecases and consider enabling them for future use in the contract.

Complete Analysis

Project Response: As mentioned, the merge functionality is a nice-to-have feature of the contract and not necessary to properly function for us. For splitting, an approval mechanism can make sense, but for merging there's additional logic around who gets the merged token after the tokens are merged together (eg: Y, Z, or X, or someone else).
Since this is more complicated logic than a traditional transfer approval, it seems best to separate it out into a new contract if it is needed in the future. In this case, Y and Z would use regular erc721 approvals to deposit the tokens into this new contract and then agree to a merge with whatever logic is needed inside the contract (eg: merge it inside the contract and hold it there for some time, and then split it afterwards).

Disclaimer:

DcentraLab Diligence (DD) has provided the code to the client as is and assumes no responsibility nor legal liability for any use client may do with the code. Any and all usage and/or deployment of the code provided by DcentraLab Diligence will be done solely by the client, at the sole discretion, responsibility, risk, and legal liability of the Client, and DD will not be held accountable or liable for any loss of funds, security exploits or incidents, or any other unintended or negative outcome that may occur in relation to the code provided by DD.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts DD to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

This report and the provided code or services as part of the SOW pertaining to this report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should it be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. DD's position is that each company and individual are responsible for their own due diligence and continuous security. DD's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by DD are subject to dependencies and are under continuing development. You agree that your access and/or use, including but not limited to any services, code, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, DcentraLab Diligence (DD) HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, DD SPECIFICALLY DISCLAIMS

ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, DD MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT / VERIFICATION REPORT, WORK PRODUCT, CODE OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

WITHOUT LIMITATION TO THE DISCLAIMER HyperCycle Contracts FOREGOING, DD PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET THE CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR-FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER DD NOR ANY OF DD'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION, CODE OR CONTENT PROVIDED THROUGH THE SERVICE. DD WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT OR CODE, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, CODE, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS," AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN THE CUSTOMER AND THE THIRD-PARTY
OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS. THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO THE CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT DD'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST DD WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS. THE REPRESENTATIONS AND WARRANTIES OF DD CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF THE CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST DD WITH RESPECT TO SUCH
REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.
FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS, CODE, OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL,
REGULATORY, OR OTHER ADVICE.

dcentralab.com/diligence

DcentraLab Diligence

Provided By DcentraLab Diligence on September 28, 2023