Cyberscope

Audit Report

HYDT Stablecoin

Jule 2023

Github	https://github.com/cc416-cr/HYDT-Protocol
Commit	a30d3d6f1e99694aa68fe9364adebeebfcbd2e72
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Review **4**
Audit Updates 4
Source Files 5
Overview **8**
HYDT 8
sHYDT 8
HYGT 9
Reserve 10
ControlResolver 11
Earn 12
Farm 13
Control 14
Testing deploy 15
Roles **16**
Governor Roles 16
Caller Roles 16
Locker Role 16
Architecture **17**
Non-Deterministic Concern 17
Access Control 18
Blockchain Data Concern 19
Stablecoin Rebalance Mechanism 20
Stimulation 20
Data Source 20
Price Formula 20
Mint Stimulation 21
Redeem Stimulation 22
Findings Breakdown **23**
Diagnostics **24**
MSC - Missing Sanity Check 26
Description 26
Recommendation 26
MVN - Misleading Variables Naming 27
Description 27
Recommendation 27
RSSF - Redundant Staking Status Functionality 28
Description 28
Recommendation 28

DKO - Delete Keyword Optimization 29
Description 29
Recommendation 29
TAP - Transfer Amount Prevalidation 30
Description 30
Recommendation 30
LSO - Lock Struct Optimization 31
Description 31
Recommendation 31
CR - Code Repetition 33
Description 33
Recommendation 33
RSW - Redundant Storage Writes 34
Description 34
Recommendation 34
WAV - Withdraw Amount Validation 35
Description 35
Recommendation 35
EUU - Ether Units Usage 36
Description 36
Recommendation 36
TUU - Time Units Usage 37
Description 37
Recommendation 37
L04 - Conformance to Solidity Naming Conventions 37
Description 38
Recommendation 39
L09 - Dead Code Elimination 40
Description 40
Recommendation 40
L13 - Divide before Multiply Operation 41
Description 41
Recommendation 41
L14 - Uninitialized Variables in Local Scope 43
Description 43
Recommendation 43
L19 - Stable Compiler Version 44
Description 44
Recommendation 44
Functions Analysis 45
Inheritance Graph 52
Flow Graph 53

Summary 54
Disclaimer 55
About Cyberscope 56

Review

Repository	https://github.com/cc416-cr/HYDT-Protocol
Commit	a30d3d6f1e99694aa68fe9364adebeebfcbd2e72

Audit Updates

Initial Audit	01 Jul 2023

Source Files

Filename	SHA256
sHYDT.sol	031e7731efaa74aaa8670f478516c4e7a23ebb3ef88674e65281e37f1b4c4193
Reserve.sol	6089fbba77c3a4dea91109d44296af2d6693d1cdc1b0775d6c322e28aa70b285
HYGT.sol	4f8c48a3b729b5771fadff74f98bbe7a2f9d381e4d9b14613a7bde272b8c1ab7
HYDT.sol	036c33b781392131a5808c6b8f034b802d08f892d02a75d847ef35dbf9dbbfa3
Farm.sol	8d4b35b8175a8fdd7ef271048f64da9a21d237ac75ad76115fdf455ee29d341e
Earn.sol	683aa7e40937bda6a650edf7c93375ca8a291915d90c6f820cf695801c982d31
ControlResolver.sol	87ce2c78365edb3e98637e86cca4883e5963876fa5162165744c33bbfe405825
Control.sol	ded89afa2a4dd57b8f919fb024722a5a490a30659cedde65793868ae60329c45
utils/OpsReady.sol	f6b9dac9b33fe7ae3f17b6f227613addfe9cac8a15dcc5bceb9d30d43906852b
utils/ERC20Permit.sol	3494d8963d1ae76bf605b49d12794409ccecf7ab5e605746a0fead52db20e60d
utils/ERC165.sol	5ba0f71e926ac7788defe62c2d25c817ad2d294d6527ef0a3d2cedfb2bea50b6
utils/EIP712.sol	af33b3254994206befe97f514dc999e79bb8656519c4696d97084a03b6a813f7

utils/Context.sol	6ee66d1e4693ec63d29393cc2c83594798475ad0eeabcb0951a35780ef003113
utils/AccessControl.sol	f591651e9d07a9652d4a22eec28a8dcded4a6a0996a362e3b9932575314c5449
libraries/Strings.sol	eed73914453d64f56137ea7ae4543ffc4f88b9195bf0fa10935793b98239103e
libraries/SignedMath.sol	e7f09613b16cf73d56bb542acda15b6cf95fddb6f99edc25d1b1f0fb4bd2d459
libraries/SafeETH.sol	a04437f39de4811921c2622269bcee2978a6b31aa1712723e89af500b792c74d
libraries/SafeERC20.sol	8aed5c25598b9f1105d0d7fc5974d7300675707610a8a8263f4c091e4be539d7
libraries/Math.sol	edb6dc365e80055e92109eeecbd0fb50566050cb5903a1366ad52f7667a46a74
libraries/ECDSA.sol	c37622b4ad062dadfaffe16389b8756528bbde27d5f86e6f530becc3fca8c06a
libraries/DataFetcher.sol	84fc8c9914442f166121f6c8d26d0740de3f5948195c0c0179bb97132053d1bc
libraries/Counters.sol	e10346b263158ef9aa46f5cbf0432b2f5221fa84f999074a93cc2fa3bd4fbb92
libraries/Address.sol	719862a65de3111feeaf4b8c44da8100461d7946672727f34c1ad5d2633e6342
interfaces/Types.sol	0ca116128e416264e310f04b99a07a38bc08529aecba484a4160db8f00d3c453
interfaces/IReserve.sol	f7cd18b4dae2098a7cf4867aed671ff2273109cceeb33ecca9e58ffd501c1bfc
interfaces/IPancakeRouter02.sol	dcbfee6a4cbf91ca36cf0d295d4130de2eef84426a64b996d01667f55a624e7d

interfaces/IPancakePair.sol	c7011f889d18d7ab68cab6707c922fda0ecd7ac5f3a88f2bd2a78b30fe51b833
interfaces/IPancakeFactory.sol	9c4d5a4084741a49adba2b8b54c3543d75f9a84217ca042c8a46b00d3b5eaa8a
interfaces/IHYGT.sol	cab0017faa61919d6eec1d4db9f8a15777941c448a33377b97f027f3772a4ed2
interfaces/IHYDT.sol	16dc507b01a4468f18f99ae72b730fe5d2a4eae016e26db7bc5440ec0124f16f
interfaces/IERC20Permit.sol	58367721c02531647b4a0d9203b124640cddd36ea2f23d7a181da5e7a368a773
interfaces/IERC20Metadata.sol	d58377f3fda62337e0b78edbb728ed77537c6bd306cff66840a060cf876956a9
interfaces/IERC20.sol	a7b8d29448cb54f1a9ba7e2a17e9cc03b62404f083ddbb90ff0a6116cf829357
interfaces/IERC165.sol	48ab1f2a907c12063745ba591bced76c1d306b6436d43ce9ce9b5ddd6c515989
interfaces/IControl.sol	4c6079a27480f3f755f6ea72ccc84ed2a56acfafe38ab5bc4c34b84232239feb
interfaces/IAccessControl.sol	7817a30761530bd700985139e6e13411e84d263d86231c6fca147ea77c5b2b5e
extensions/ERC20.sol	a353c7109e40f2b86f43ea47f92857d2364722be6f86299a69ee60ef4f9b9086

Overview

HYDT

The contract implements basic minting and burning functionality for the HYDT token, with role-based access control to restrict certain operations.

sHYDT

The sHYDT contract implements an ERC20 token called "sHYDT" (Staked High Yield Dollar Stable Token).

HYGT

The HYGT contract implements the HYGT (High Yield Dollar Governance Token) token.

The contract initializes with the constructor, which mints an initial supply of HYGT tokens and assigns roles to specific addresses.

The contract includes functions for unlocking vested tokens, such as unlock, which allows lockers to mint their tokens based on a predetermined schedule.

It includes functions for minting and burning tokens, which are restricted to addresses with the Caller role.

The contract implements a voting mechanism, where token holders can delegate their votes to other addresses using the delegate function.

The getCurrentVotes and getPriorVotes functions provide information about the current and prior voting balances for an address.

The contract includes internal functions for managing vote delegation and checkpoint updates.

Reserve

The Reserve contract manages the storage and withdrawal of BNB tokens.

The contract includes a fallback function (receive) that accepts BNB transfers. It calculates the total reserve value based on the received BNB balance and emits an In event.

The withdraw function allows an authorized caller to withdraw a specified amount of BNB from the contract to their address. It transfers the BNB using the SafeETH.safeTransferETH function and emits an Out event.

Overall, this contract serves as a reserve that accepts incoming BNB transfers and allows authorized callers to withdraw BNB from the reserve. It also calculates and emits events for the total reserve value based on the received BNB balance.

ControlResolver

The ControlResolver contract checks the execution status based on predefined conditions and interacts with the Control contract to execute specific functions when the conditions are met.

The contract contains a checker function that checks the execution status. Within the checker function, the current price, mint progress count, redeem progress count, last executed mint timestamp, and last executed redeem timestamp are retrieved from the CONTROL contract.

The checker function evaluates conditions for executing the mint or redeems functions based on the price and the time elapsed since the last execution.

If the conditions for executing mint or redeem are met, the execute function of the CONTROL contract is encoded as a function call and returned as the execPayload.

If the conditions are not met, information about the price, mint/redeem last execution time, and progress counts are returned as the execPayload.

Earn

The Earn contract allows users to stake their HYDT (High Yield Dollar Stable Token) and earn rewards in both HYDT and HYGT (High Yield Governance Token).

The contract includes events to emit relevant information when users stake, claim payouts, or unstake.

The contract provides functions to update the allocation points for pools, update reward variables, and retrieve pending rewards and payouts for users.

Users can stake their HYDT tokens using the stake function, specifying the amount and stake type. A fee is deducted, and the remaining amount is transferred to the contract and converted to sHYDT tokens.

Users can claim their pending rewards and payouts using the claimPayout function, providing the index of the staking position.

Farm

The Farm contract implements staking LP tokens and earning HYGT tokens as rewards.

The contract provides functions for adding pools, updating allocation points, and updating reward variables for all pools.

There are functions to get pending HYGT rewards for a user in each pool, mass update pools to update reward variables for all pools, and update pool-specific reward variables.

The contract includes functions for depositing LP tokens to receive HYGT rewards, withdrawing LP tokens from a pool, and emergency withdrawing LP tokens without caring about rewards.

Control

The Control contract implements various functionalities related to the control and operation of a decentralized application (DApp). The Control encapsulates the fundamental functionality of the ecosystem, being responsible for executing the mint and redeem mechanisms to rebalance the stablecoin.

It defines several state variables including role constants, time duration variables, addresses of external contracts, initial minting limits, price bounds, and instances of other contracts.

The contract emits various events to notify important state changes.

The contract includes various external and internal functions for controlling the contract's behavior. These functions handle tasks such as updating slippage tolerance, updating the ops ready state, delegating token approvals, getting initial minting information, getting the current HYDT price, performing initial minting, executing operations to maintain the peg, and more.

Testing deploy

Contract Name	Explorer
HYDT	https://testnet.bscscan.com/address/0xed371e9EE0d89dB1fB9C7477E3BDb29f3E67F350
HYGT	https://testnet.bscscan.com/address/0xfc7AB1570C7142ece89DC2d0995128cFD5b0Fe35
sHYDT	https://testnet.bscscan.com/address/0xe94fEE13B21Ca79F892EF42fFb5B8D71f21A1b66
Control	https://testnet.bscscan.com/address/0x7fd2A4baFb556c9D385A6c515720c6B59Ead0eA1
ControlResolver	https://testnet.bscscan.com/address/0xB3b7820AAd173033BFdb758bD8475948Ae9bA180
Earn	https://testnet.bscscan.com/address/0x18b4CbE1A94F68a7d430CE724C3EadEdfdA3f25a
Farm	https://testnet.bscscan.com/address/0x09aDF3c05633F2B6e75376FDD4acC2ceA719e272
Reserve	https://testnet.bscscan.com/address/0x3b78bbfdA52228bBf7D1061667A74007Bf8A1479

Roles

The ecosystem roles consist of three roles.

Governor Roles

The Governor role is responsible for configuring ecosystem parameters.

Caller Roles

The Caller role is responsible for calling functions that make changes to the ecosystem.

Locker Role

The Locker role is responsible for unlocking vested tokens.

Architecture

Non-Deterministic Concern

It is crucial to prioritize a deterministic implementation. By ensuring that the system behaves consistently and predictably, we can enhance user trust and confidence. The contract's deterministic performance may be impacted due to its reliance on an external service. As the ecosystem utilizes this external service, it introduces an element of uncertainty that can affect the deterministic behavior of the contract.

One approach to optimize and achieve determinism within the ecosystem is to transfer the stablecoin rebalance functionality to the transfer transaction. By integrating the rebalance process directly into the transfer transaction, the ecosystem gains the advantage of greater efficiency and predictability. This implementation ensures that every transfer of stablecoins is accompanied by an automatic rebalancing action, maintaining the stability of the ecosystem. This optimization leads to a more streamlined and reliable operation, reducing the need for separate rebalancing mechanisms and providing a more coherent and synchronized experience for users. Additionally, this approach enhances the overall robustness of the ecosystem, ensuring smooth and deterministic rebalancing, regardless of external factors or market fluctuations.

Access Control

To streamline the roles and access control functionality within the ecosystem, a recommended approach is to consider moving them to a library contract. Consolidating all the roles and access control logic into a separate library contract offers several advantages. Firstly, it promotes code reusability and modularity, allowing for cleaner and more maintainable contract architecture. Secondly, centralizing these functionalities in a library contract enables easier upgrades and modifications, as changes made to the library contract propagate throughout the ecosystem. Moreover, by decoupling the roles and access control logic from the main contract, potential security risks and complexities can be reduced. This approach enhances code readability, simplifies contract development, and provides a robust foundation for managing roles and access control throughout the ecosystem.

The presence of multiple instances of the same roles is observed in a significant number of contracts. For instance,

```
bytes32 public constant GOVERNOR_ROLE =
keccak256(abi.encodePacked("Governor"));
bytes32 public constant CALLER_ROLE =
keccak256(abi.encodePacked("Caller"));
```

Blockchain Data Concern

In several instances, the contract makes use of the same constant variables across multiple contracts, resulting in duplication of code and potential maintenance challenges. To address this, it would be beneficial to consider moving these common variables to a utility contract. By centralizing these shared variables in a utility contract, we can promote code reusability, enhance maintainability, and reduce redundancy across multiple contracts.

```
address public constant PANCAKE_FACTORY =
0xcA143Ce32Fe78f1f7019d7d551a6402fC5350c73;
address public constant WBNB =
0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c;
address public constant USDT =
0x55d398326f99059fF775485246999027B3197955;
```

Furthermore, although the contract utilizes a router, it currently manually adds information to the contract instead of retrieving it from the router. This approach may lead to inconsistencies and manual errors.

Stablecoin Rebalance Mechanism

Upon reviewing the rebalance architecture of the stablecoin, we observed that the process relies on both the current price of the stablecoin and the reserve balance of the ecosystem. Furthermore, to prevent significant fluctuations in stablecoin prices, the rebalance functionality is capped with a percentage of the liquidity amount.

Stimulation

The purpose of the simulation was to observe how the rebalance functionality behaves through multiple mints and redeems. After each mint or redeem the price was calculated. Furthermore, the Mint and Redeem formulas were simulated with 0% slippage.

By conducting simulations on the mint and redeem functionalities, we observed that the mint function tends to exhibit more aggressive price changes compared to the redeem function, as indicated in the chart.

Data Source

Data was forked from BSC Blockchain for the simulation, meaning that prices and pair supplies were retrieved directly from the blockchain.

Price Formula

The formula calculates the price of the token HYDT in terms of the BUSD token

1. Calculate the price of 1 HYDT token in terms of WBNB (Wrapped Binance Coin): `HYDTBNBprice = 1 HYDT * HYDTpair.WBNBBalance / HYDTpair.BUSDBalance`
2. Calculate the price of 1 HYDT token in terms of BUSD: `price = (1 HYDTBNBprice * BUSDpair.BUSDBalance) / BUSDpair.WBNBBalance`

```
price = (((1 HYDT * HYDTpair.WBNBBalance) / HYDTpair.BUSDBalance) *
BUSDpair.BUSDBalance) / BUSDpair.WBNBBalance
```

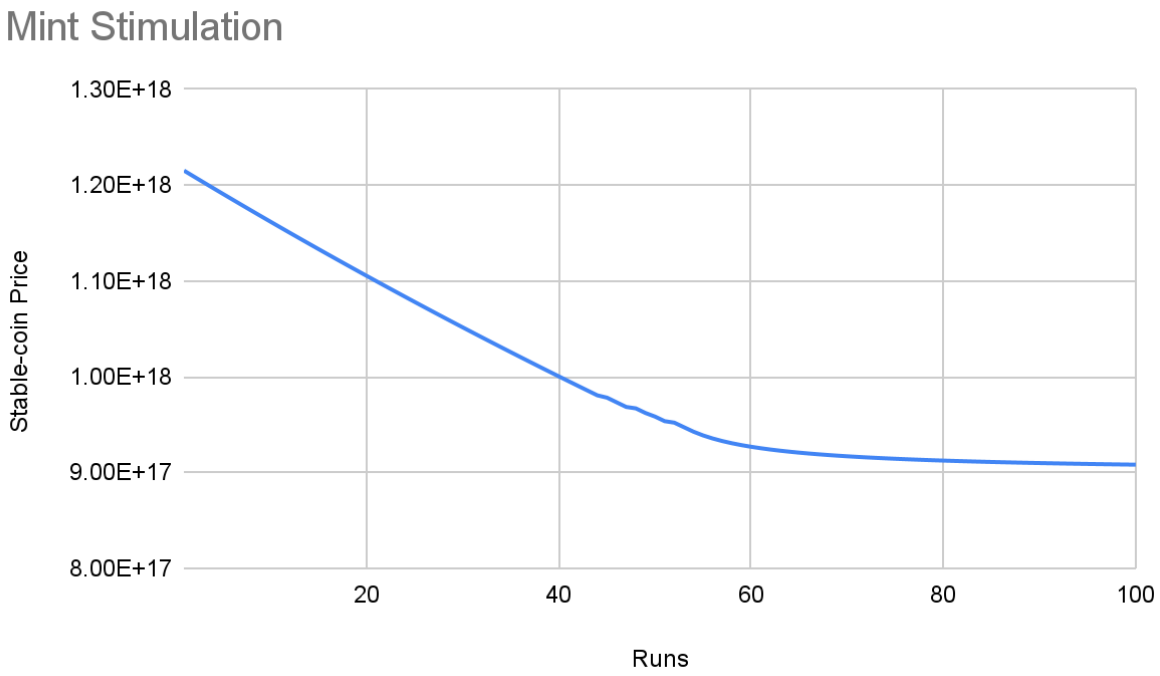
Mint Stimulation

Illustrates the price change in relation to mint executions.

The mint amount was calculated with the following formula :

```
Mint amount = (Current price - 0.9)^2 * Reservebalance * 0.04
```

Mint stimulation starting price: 1.20 $
Mint stimulation ending price: 0.9 $



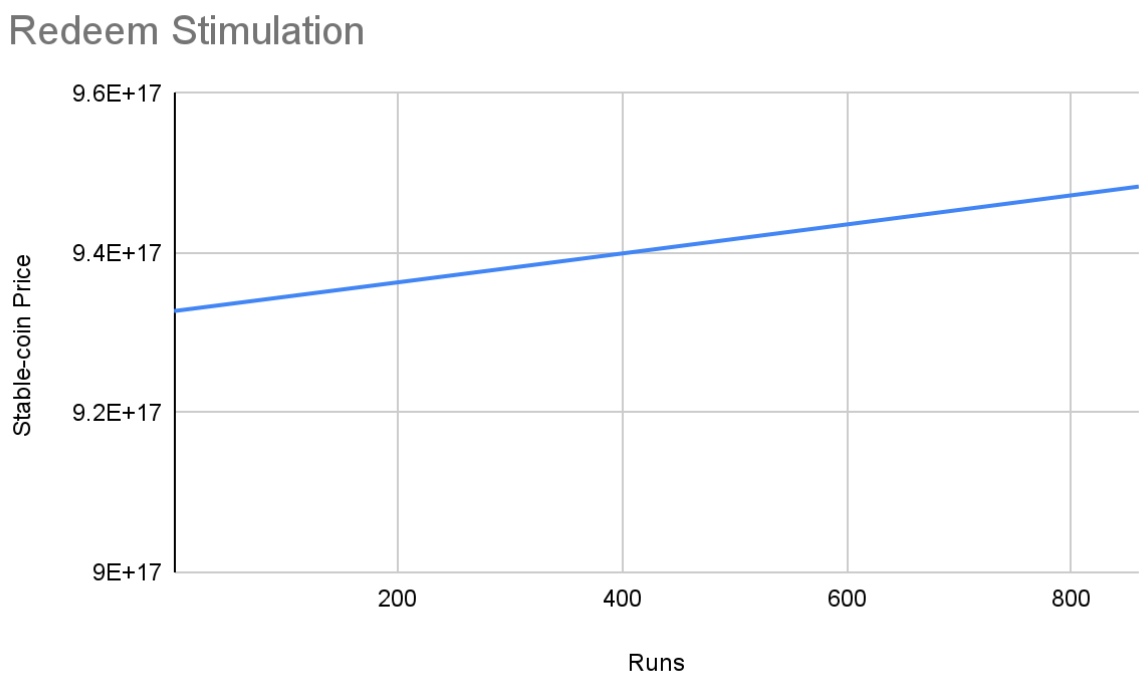
Redeem Stimulation

Illustrates the price change in relation to multiple redeem executions.

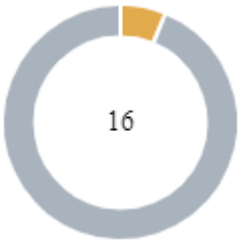
The redeem amount was calculated with the following formula :

```
Redeem amount = (1.1 - Current price)^2 * Reservebalance * 0.004
```

Redeem stimulation starting price: 0.93 $
Redeem stimulation ending price: 0.95 $



Findings Breakdown



Critical	0
Medium	1
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	15	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	MSC	Missing Sanity Check	Unresolved
Minor / Informative	MVN	Misleading Variables Naming	Unresolved
Minor / Informative	RSSF	Redundant Staking Status Functionality	Unresolved
Minor / Informative	DKO	Delete Keyword Optimization	Unresolved
Minor / Informative	TAP	Transfer Amount Prevalidation	Unresolved
Minor / Informative	LSO	Lock Struct Optimization	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	WAV	Withdraw Amount Validation	Unresolved
Minor / Informative	EUU	Ether Units Usage	Unresolved
Minor / Informative	TUU	Time Units Usage	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L13	Divide before Multiply Operation	Unresolved

●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L19	Stable Compiler Version	Unresolved

MSC - Missing Sanity Check

Criticality	Medium
Location	Earn.sol#L266,292,315,381,414Farm.sol#L210,228,252,277
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The arguments `pid`, `user`, `index`, and `amount` are not properly sanitized.

```
function getPending(uint256 pid, address user) public view returns
(uint256) { ... }
function getPending(address user, uint256 index) public view returns
(uint256) { ... }
function deposit(uint256 pid, uint256 amount) external { ... }
function withdraw(uint256 pid, uint256 amount) external { ...}
function stake(uint256 amount, uint8 stakeType) external { ... }
function claimPayout(uint256 index) external { ... }
...
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

- The variable `index` should be a valid index of the pool index.
- The variable `user` should be a valid address and it must be a valid user.
- The variable `pid` should be a valid pool id.
- The variable `amount` should be greater than zero.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	HYGT.sol#L38
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

The contract does not have any tokens in the contract. It mints the token on the `unlock` function.

```
struct Lock {
    bool status;
    uint256 baseAmount;
    uint256 unlockedAmount;
    uint256 totalAmount;
    uint256 startTime;
    uint256 intervalCounter;
    uint256 totalIntervals;
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

RSSF - Redundant Staking Status Functionality

Criticality	Minor / Informative
Location	Earn.sol#L414
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract is utilizing the `staking.status` variable in the contract implementation. The staking status variable is used only to determine whether the staking element exists. There are variables that can be reused to determine whether the staking element exists. Hence, the `staking.status` is redundant.

```
function stake(uint256 amount, uint8 stakeType) external {
    ...
    staking.status = true;

function claimPayout(uint256 index) external {
    require(staking.status, "Earn: invalid staking");
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. It is recommended to reuse existing variables to determine if a staking element exists in the ecosystem. For instance, The contract could utilize the `staking.lastClaimTime`

```
require(staking.lastClaimTime>0);
```

DKO - Delete Keyword Optimization

Criticality	Minor / Informative
Location	Farm.sol#L277
Status	Unresolved

Description

The contract resets variables to the default state by setting the initial values. Setting values to state variables increases the gas cost.

```
function emergencyWithdraw(uint256 pid) public {
    ...
    userData.amount = 0;
    userData.rewardDebt = 0;
}
```

Recommendation

The team is advised to use the `delete` keyword instead of setting variables. This can be more efficient than setting the variable to a new value, using delete can reduce the gas cost associated with storing data on the blockchain.

TAP - Transfer Amount Prevalidation

Criticality	Minor / Informative
Location	Earn.sol#L381Control.sol#L275
Status	Unresolved

Description

The current implementation of the contract does not prevalidate whether sufficient tokens are available to perform transactions, which can lead to transaction failures or unexpected behavior. For example, if a user attempts to transfer tokens that are not available, the transaction will fail and potentially leave the contract in an inconsistent state.

```
SafeERC20.safeTransferFrom(HYDT, _msgSender(), TREASURY, fee);
SafeETH.safeTransferETH(address(RESERVE), msg.value);
```

Recommendation

It is recommended to implement prevalidation checks before any transaction. To ensure that there are sufficient tokens available for any transaction. By performing prevalidation checks, the contract will be more reliable and less prone to errors or vulnerabilities.

LSO - Lock Struct Optimization

Criticality	Minor / Informative
Location	HYGT.sol#L38
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract utilizes the variable `baseAmount` to correct the solidity division precision. This code segment could be optimized.

```
struct Lock {
    bool status;
    uint256 baseAmount;
    uint256 unlockedAmount;
    uint256 totalAmount;
    uint256 startTime;
    uint256 intervalCounter;
    uint256 totalIntervals;
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

The contract could eliminate the additional variable `baseAmount` and the associated functionality. Instead, it can calculate the `baseAmount` by subtracting the `lock.unlockedAmount` from the `lock.totalAmount`, and then on the last interval aggregating it with the unlock amount. This approach would eliminate the need for an extra variable.

```
if (lock.intervalCounter == lock.totalIntervals) {
    lock.status = false;
    lock.unlockedAmount += lock.totalAmount-lock.unlockedAmount;
}
```

CR - Code Repetition

Criticality	Minor / Informative
Location	Control.sol#L224,256
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
if (block.timestamp <= _initialMints.endTime && block.timestamp >
_dailyInitialMints.endTime) {
    uint256 numberOfDays = (block.timestamp -
_dailyInitialMints.startTime) / ONE_DAY_TIME;
    startTime = _dailyInitialMints.startTime + (numberOfDays *
ONE_DAY_TIME);
    endTime = _dailyInitialMints.endTime + (numberOfDays *
ONE_DAY_TIME);
    amountUSD = 0;
}

if (block.timestamp > dailyInitialMints.endTime) {
    uint256 numberOfDays = (block.timestamp -
dailyInitialMints.startTime) / ONE_DAY_TIME;
    dailyInitialMints.startTime += numberOfDays * ONE_DAY_TIME;
    dailyInitialMints.endTime += numberOfDays * ONE_DAY_TIME;
    dailyInitialMints.amount = 0;
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Farm.sol#L149Earn.sol#L187
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes.

The variable `allocPoint` could be the same.

```
function updateAllocation(uint256 pid, uint256 allocPoint, bool
withUpdate) external onlyRole(GOVERNOR_ROLE) {
    if (withUpdate) {
        massUpdatePools();
    }
    totalAllocPoint = (totalAllocPoint - poolInfo[pid].allocPoint) +
allocPoint;
    uint256 oldAllocPoint = poolInfo[pid].allocPoint;
    poolInfo[pid].allocPoint = allocPoint;

    emit UpdateAllocation(pid, allocPoint, oldAllocPoint);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

WAV - Withdraw Amount Validation

Criticality	Minor / Informative
Location	Reserve.sol#L74
Status	Unresolved

Description

The contract is missing withdraw amount validation in the `withdraw` function. The absence of validation may lead to revert of the method.

```
function withdraw(uint256 amount) external onlyRole(CALLER_ROLE) {
    SafeETH.safeTransferETH(_msgSender(), amount);
    uint256 totalReserveBNB = address(this).balance;
    uint256 totalReserve = DataFetcher.quote(PANCAKE_FACTORY,
totalReserveBNB, WBNB, USDT);

    emit Out(_msgSender(), amount, totalReserveBNB, totalReserve);
}
```

Recommendation

It is recommended to implement a sufficient withdraw amount validation mechanisms. A recommended approach could be to verify that the withdrawal amount is sufficient to perform the transaction.

EUU - Ether Units Usage

Criticality	Minor / Informative
Location	Control.sol#L52,53,239
Status	Unresolved

Description

The contract is using arbitrary numbers to form ether-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 private constant PRICE_UPPER_BOUND = 1.02 * 1e18;
uint256 private constant PRICE_LOWER_BOUND = 0.98 * 1e18;

uint256 amountIn = 1 * 1e18;
```

Recommendation

It is a good practice to use the ether units reserved keyword `ethers` to process ether-related calculations.

It's important to note that these ether unit is simply a shorthand notation for representing ether, and do not have any effect on the actual passage of ether or the execution of the contract. The ether unit id simply a convenience for expressing ethers in a more human-readable form. For instance,

```
uint128 private constant PRICE_UPPER_BOUND = 1.02 ether;
uint128 private constant PRICE_LOWER_BOUND = 0.98 * ether;
```

TUU - Time Units Usage

Criticality	Minor / Informative
Location	ControlResolver.sol#L15,16HYGT.sol#L19Farm.sol#L19,31Earn.sol#L20,21,136,139Control.sol#L25,26,27
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint128 private constant FOUR_HOURS_TIME = 14400;
uint128 private constant FIVE_MINUTES_TIME = 300;

uint128 private constant ONE_MONTH_TIME = 2592000;

uint128 private constant ONE_DAY_TIME = 86400;
lockPeriods = [7776000, 15552000, 31536000];
HYGTPerSecond = 0.666666666666666667 * 1e18;

uint256 averageBlockTime = 3;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative

Location	sHYDT.sol#L11HYGT.sol#L27Farm.sol#L22,25Earn.sol#L24,26,30,42ControlResolver.sol#L23Control.sol#L56,58
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
contract sHYDT is ERC20, IHYDT, AccessControl {

    /* ========== STATE VARIABLES ========== */

    bytes32 public constant CALLER_ROLE =
keccak256(abi.encodePacked("Caller"));

...
    function burnFrom(address from, uint256 amount) external override
onlyRole(CALLER_ROLE) returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _burn(from, amount);
        return true;
    }
}

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	utils/AccessControl.sol#L205,214libraries/DataFetcher.sol#L44,60
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _setupRole(bytes32 role, address account) internal virtual {
        _grantRole(role, account);
    }

function _setRoleAdmin(bytes32 role, bytes32 adminRole) internal
virtual {
        bytes32 previousAdminRole = getRoleAdmin(role);
        _roles[role].adminRole = adminRole;
        emit RoleAdminChanged(role, previousAdminRole, adminRole);
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	HYGT.sol#L190,191,194Farm.sol#L190,191,218,219Earn.sol#L228,229,304,305,367,371,372,426,427Control.sol#L224,225,226,256,257,258,321,322,328,362,396,397,401,403,441
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 numberOfIntervals = (block.timestamp - lastUnlockTime) /
ONE_MONTH_TIME
uint256 unlockAmount = (lock.totalAmount / lock.totalIntervals) *
numberOfIntervals
numberOfIntervals = numberOfIntervals > maxIntervals ? maxIntervals :
numberOfIntervals

uint256 HYGTReward = (numberOfBlocks * HYGTPerBlock * pool.allocPoint)
/ totalAllocPoint
pool.accHYGTPerShare += ((HYGTReward * 1e12) / lpSupply)

uint256 HYGTReward = (numberOfSeconds * HYGTPerSecond *
pool.allocPoint) / totalAllocPoint
pool.accHYGTPerShare += (HYGTReward * 1e12) / pool.stakeSupply

uint256 baseValue = (((price - (0.9 * 1e18)) ** 2) * amountReserve *
(0.04 * 1e2)) / 1e38
uint256 firstValue = (baseValue * mintProgressCount) / 1e18
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses

to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Farm.sol#L133Earn.sol#L168,279,293,316,348,362,394Control.sol#L336,410,413
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
PoolInfo memory pool
uint256 totalPending
uint256 pending
uint256 totalDailyPayout
uint256 totalPayout
uint256 payout
Staking memory staking
bool check
uint256 amountBurnHYDT
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	utils/AccessControl.sol#L4interfaces/IAccessControl.sol#L4
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
sHYDT	Implementation	ERC20, IHYDT, AccessControl		
		Public	✓	ERC20
	initialize	External	✓	-
	mint	External	✓	onlyRole
	burn	External	✓	-
	burnFrom	External	✓	onlyRole
Reserve	Implementation	AccessControl		
		Public	✓	-
	initialize	External	✓	-
		External	Payable	-
	withdraw	External	✓	onlyRole
HYGT	Implementation	IHYGT, AccessControl, ERC20Permit		
		Public	✓	ERC20 ERC20Permit

	initialize	External	✓	-
	maxTotalSupply	External		-
	getCurrentVotes	External		-
	getPriorVotes	External		-
	unlock	External	✓	onlyRole
	mint	External	✓	onlyRole
	burn	External	✓	-
	burnFrom	External	✓	onlyRole
	delegate	External	✓	-
	_delegate	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_moveDelegates	Internal	✓	
	_writeCheckpoint	Internal	✓	
HYDT	Implementation	IHYDT, AccessControl, ERC20Permit		
		Public	✓	ERC20 ERC20Permit
	initialize	External	✓	-
	mint	External	✓	onlyRole
	burn	External	✓	-
	burnFrom	External	✓	onlyRole
Farm	Implementation	AccessControl		

		Public	✓	-
	initialize	External	✓	-
	poolLength	External		-
	addPool	External	✓	onlyRole
	_addPool	Internal	✓	
	updateAllocation	External	✓	onlyRole
	massUpdatePools	Public	✓	-
	updatePool	Public	✓	-
	getPendingBatch	External		-
	getPending	Public		-
	deposit	External	✓	-
	withdraw	External	✓	-
	emergencyWithdraw	Public	✓	-
Earn	Implementation	AccessControl		
		Public	✓	-
	initialize	External	✓	-
	poolLength	External		-
	allPoolInfo	External		-
	_addPool	Internal	✓	
	updateAllocationWithUpdate	Public	✓	onlyRole
	massUpdatePools	Public	✓	-
	updatePool	Public	✓	-

	_binarySearchShare	Internal		
	getPendingBatch	External		-
	getPendingType	External		-
	getPending	Public		-
	getDailyPayoutBatch	External		-
	getPayoutBatch	External		-
	getPayoutType	External		-
	getPayout	Public		-
	stake	External	✓	-
	claimPayout	External	✓	-
ControlResolver	Implementation	Context		
		Public	✓	-
	initialize	External	✓	-
	checker	External		-
Control	Implementation	AccessControl, OpsReady		
		Public	✓	-
	initialize	External	✓	-
		External	Payable	-
	updateSlippageTolerance	External	✓	onlyRole
	updateOpsReadyState	External	✓	onlyRole
	delegateApprove	External	✓	onlyRole

	_delegateApprove	Internal	✓	
	getInitialMints	External		-
	getDailyInitialMints	External		-
	getCurrentPrice	Public		-
	initialMint	External	Payable	-
	execute	External	✓	onlyRole
	_mint	Internal	✓	
	_redeem	Internal	✓	
AccessControl	Implementation	Context, IAccessControl, ERC165		
	supportsInterface	Public		-
	hasRole	Public		-
	_checkRole	Internal		
	_checkRole	Internal		
	getRoleAdmin	Public		-
	grantRole	Public	✓	onlyRole
	revokeRole	Public	✓	onlyRole
	renounceRole	Public	✓	-
	_setupRole	Internal	✓	
	_setRoleAdmin	Internal	✓	
	_grantRole	Internal	✓	
	_revokeRole	Internal	✓	

DataFetcher	Library			
	pairFor	Internal		
	getReserves	Internal		
	quote	Internal		
	quoteBatch	Internal		
	quoteRouted	Internal		
IReserve	Interface			
	withdraw	External	✓	-
IHYGT	Interface	IERC20		
	maxTotalSupply	External		-
	mint	External	✓	-
	burn	External	✓	-
	burnFrom	External	✓	-
	delegate	External	✓	-
	getCurrentVotes	External		-
	getPriorVotes	External		-
IHYDT	Interface	IERC20		
	mint	External	✓	-
	burn	External	✓	-
	burnFrom	External	✓	-

IControl	Interface			
	mintProgressCount	External		-
	redeemProgressCount	External		-
	lastExecutedMint	External		-
	lastExecutedRedeem	External		-
	delegateApprove	External	✓	-
	getDailyInitialMints	External		-
	getInitialMints	External		-
	initialMint	External	Payable	-
	getCurrentPrice	External		-
	execute	External	✓	-
IAccessControl	Interface			
	hasRole	External		-
	getRoleAdmin	External		-
	grantRole	External	✓	-
	revokeRole	External	✓	-
	renounceRole	External	✓	-

Inheritance Graph

Flow Graph

Summary

HYDT Stablecoin contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. 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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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 Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services 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.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io