THE ARCADIA GROUP

Audit of Herdius DEX Smart Contracts

a report of findings by

Yassine Amor

innovative fortuna iuvat

November 23rd, 2020

Table of Contents

Document Info

Client	Herdius
Title	Smart Contract Audit
Auditors	Yassine Amor
Reviewed By	Joel Farris
Approved By	Rasikh Morani

Contact

For more information on this report, contact The Arcadia Media Group Inc.

Rasikh Morani
(972) 543-3886
rasikh@arcadiamgroup.com
https://t.me/thearcadiagroup

Executive Summary

A Representative Party of Herdius ("Herdius") engaged The Arcadia Group ("Arcadia"), a software development, research and security company, to conduct a review of the following smart contracts on the herdius-defi repo at Commit #73485855fb5b0245c4e6576dbd2213cc7f7ad235.

RewardsEscrow.sol

HToken.sol

HerdiusMarket.sol

HerdiusFactory.sol

HerdiusDeployProxy.sol

IRewardsEscrow.sol

IHToken.sol

IHerdiusMarket.sol

IHerdiusFactory.sol

IHerdiusDeployProxy.sol

IERC20.sol

Arcadia completed the review using various methods primarily consisting of dynamic and static analysis. This process included a line by line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

Findings

HerdiusFactory.sol

1. Variable Should Be Constant

- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: HerdiusFactory.sol
- Category: Informational
- Finding Type: Static
- Line: 84

In the `createMarket` function of the HerdiusFactory.sol contract, a variable factor is declared and initialized but never re-assigned a value.

```
uint factor = 1e18;
```

Action recommended: Make the factor variable a constant.

2. Gas Optimization

- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: HerdiusFactory.sol
- Category: Gas
- CWE-1164
- Finding Type: Static
- Lines: 89, 110,236

Line 89: In the `createMarket` function of the HerdiusFactory.sol contract, a loop is created to check that the asset address is not a zero address. Since the transaction reverts as soon as a zero address is found, there’s no need to loop through the entire assets array before reverting.

```
for (uint112 i = 0; i < assets.length; i++) {
            _weights[i] = uint112(weights[i]);
            if (assets[i] == address(0)) {
                isZeroAddress = true;
            }
        }
         require(isZeroAddress == false, "One of the provided addresses is
0 address.");
```

Action recommended: Check zero address inside for loop.

```
for (uint112 i = 0; i < assets.length; i++) {
             require(assets[i]!=address(0),'One of the provided addresses
is 0 address');
            _weights[i] = uint112(weights[i]);
        }
```

Line 110: In the *createMarket* function of the HerdiusFactory.sol contract, the *market* variable containing the created market's address is returned but is never used.

```
 return market;
```

Action recommended: Remove the return statement if there's no plan to use it.

Line 129: In the *accountLiquidity* function of the HerdiusFactory.sol contract, two variables *(uint,uint)* are returned but the first returned value is never used.

```
function accountLiquidity(address market, address borrower, uint assetId,
uint _toBorrow, uint _toRedeem) internal view returns (uint, uint)
```

Action recommended: Remove the first returned value.

Line 235: In the *penaltyCheck* function of the HerdiusFactory.sol contract, the *_marketBlock* variable which contains the block number is declared and assigned a value but is never used. Instead, a new variable *_currentBlock* is declared and used in the call to *calculatePenalty*.

```
uint _currentBlock = block.number;
uint _marketBlock;
(_values,_marketBlock)=IHerdiusMarket(allMarkets[i]).getPenaltyAccount(acc
ount);
(, uint feePaid, uint shortFall) = calculatePenalty(_values,
_currentBlock);
```

[redacted]

Action recommended: Use the `_marketBlock` variable as an argument for `_calculatePenalty` instead of `_currentBlock`

3. Missing Documentation

- Severity: Notice
- Likelihood: Notice
- Impact: Notice
- Target: HerdiusFactory.sol
- Category: Code Clarity
- Finding Type: Static

In the HerdiusFactory.sol, some functions containing complex logic are missing documentation.

- accountLiquidity
- distributeFees
- lendingOverview

Action recommended: Be sure to document every function, independently of the function's complexity.

HToken.sol

1. Logic Separation

- Severity: Notice
- Likelihood: Notice
- Impact: Notice
- Target: HToken.sol
- Category: Code Clarity
- Finding Type: Static

The HToken.sol contract contains both the ERC-20 logic and the Herdius-specific logic. It is better for code readability and clarity to create two separate files separating the logic.

2. Minting Can Result In Burning Tokens

- Severity: High
- Likelihood: High
- Impact: High
- Target: HToken.sol
- Category: High
- Finding Type: Static
- Line: 130

In the `mint` function of the HToken.sol contract, the _mint function is called before checking the receiver of the tokens. Having such code means that the tokens will get burned by calling the first _mint function with a receiver equal to a zero address.

```
function mint(address to, uint value) nonReentrant external {
        require(_minters.has(msg.sender), "DOES_NOT_HAVE_MINTER_ROLE");
        _mint(to, value);
        if (to == address(0)) {
            _mint(msg.sender, value);
        }
    }
```

Action recommended: Remove the _*mint* function call in Line 132 and add an *else* statement.

```
function mint(address to, uint value) nonReentrant external {
        require(_minters.has(msg.sender), "DOES_NOT_HAVE_MINTER_ROLE");
        if (to == address(0)) {
            _mint(msg.sender, value);
        } else{
            _mint(to, value);
        }
    }
```

3. Anyone Can Mint

- Severity: High
- Likelihood: High
- Impact: High
- Target: HToken.sol
- Category: High
- Finding Type: Static

- Line: 130

The intended functionality of the mint function is to only allow the markets to mint tokens. This is not checked in the code.

```
function mint(address to, uint value) nonReentrant external {
        require(_minters.has(msg.sender), "DOES_NOT_HAVE_MINTER_ROLE");
        _mint(to, value);
        if (to == address(0)) {
            _mint(msg.sender, value);
        }
    }
```

```
function addMinter(address toAdd) external onlyOwner {
        _minters.add(toAdd);
        emit MinterAdded(toAdd);
    }
```

Action recommended:

1. Set the factory as the only address that can add minters by creating a modifier or a function and remove the *onlyOwner* modifier.
2. In the createMarket function of the *HerdiusFactory* contract, when a market is created, the factory should call the addMinter function of the *HToken* contract with the new market address as the *toAdd* param.

This way, the only minters authorized to mint are the markets created by the factory which satisfies the intended functionality.

HerdiusMarket.sol

1. Unused variable

- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: HerdiusMarket.sol
- Category: Informational
- CWE-1164
- Finding Type: Static
- Line: 27

In the HerdiusMarket.sol contract, there is an unused variable which may be used in the future.

```
uint[] private emptyArray;
```

Action recommended: Remove unused variables.

2. Double Protection against Reentrancy

- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: HerdiusMarket.sol
- Category: Informational
- CWE-1164
- Finding Type: Static
- Lines: 421,500,584

In the HerdiusMarket.sol contract, functions **mint**, **burn** and **swap** utilize both the nonReentrant and the lock modifier. Both modifiers do the same thing, only one of them is required to protect against reentrancy.

Line 421:

```
function mint(address to, address[] calldata _assets) external
nonReentrant lock returns (uint liquidity) {
```

Line 500:

```
function burn(uint hAmount, address asset) external nonReentrant lock
returns (uint) {
```

Line 584:

```
function swap(address tokenA, address tokenB, uint amountOut, uint
amountIn, address to) external nonReentrant lock {
```

Action recommended: Remove the duplicate modifier.

3. Code Readability

- Severity: Low
- Likelihood: Low

- Impact: Low
- Target: HerdiusMarket.sol
- Category: Informational
- CWE-1164
- Finding Type: Static
- Lines: 133,138,143,149,154,204,216,284,908

Repeated require statements checking code should have a modifier or a function in the contract. Specifically, `require(msg.sender == factory, 'not factory');` is repeated 9 times.

```
 function changeFactory(address to) external {
        require(msg.sender == factory, 'not factory');
        factory = to;
    }
     // changes the lendBorrowlock variable, callable by factory which in
turn can only be called by the owner.
    function changeLendBorrowLock(uint value) external {
        require(msg.sender == factory, 'Not factory');
        lendBorrowUnlocked = value;
    }

    function changeFeePercentage(uint value) external {
        require(msg.sender == factory, 'Not factory');
        feePercentage = value;
    }...
```

Action recommended: Implement a function or a modifier to check for the `msg.sender == factory` condition.

4. Integer Overflow

- Severity: High
- Likelihood: High
- Impact: High
- Target: HerdiusMarket.sol
- Category: Security
- CWE-682
- Finding Type: Static
- Lines: 446

In the HerdiusMarket.sol contract, liquidity is calculated using the product of _amounts[j] and amounts[j+1] without checking for overflow conditions.

```
liquidity        +=        Math.sqrt(_amounts[j]        *        _amounts[j
+1]).sub(MINIMUM_LIQUIDITY);
```

Action recommended: Use SafeMath for arithmetic operations.

RewardsEscrow.sol

1. State Visibility

- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: RewardsEscrow.sol
- Category: Informational
- CWE-710
- Finding Type: Static
- Lines: 24,25,26

Variable visibility is not set for multiple variables in the RewardsEscrow.sol contract. A public variable is easily accessible to users of the contract, and while no specific issues were found with the specific variables identified, it is best practice to utilize the most restrictive visibility as possible, unless it is specifically needing to be public, in which case specifically set it to public.

```
    mapping(address => mapping(address => uint256))
accountFeeBalances; // maps account addres => assetAddress => amount
    mapping(address => uint256) lastFees; // maps accounts to their
fees last time they were awarded
    mapping(address => uint256) assetSupply; // maps asset/token
address to the supply the escrow owns
```

Action Recommended: In future versions of the RewardsEscrow.sol contract, specifically set the visibility status for each variable, even if it’s intended to be public. While this is not an issue that requires immediate attention, it is important to do this for code quality, health and review purposes.

2. Unused variables

- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: RewardsEscrow.sol
- Category: Informational
- CWE-1164
- Finding Type: Static
- Lines: 25,63

In the RewardsEscrow.sol contract, there are unused variables which may be used in the future, it is recommended at this time on future deployments if the variable is still unused, to remove it as unutilized code.

Line 25: lastFees variable is unused.

```
mapping(address => uint256) lastFees; // maps accounts to their fees last
time they were awarded
```

Line 63: fees variable is unused.

```
function distributeEscrowFees(address[] calldata assets, address[]
calldata accounts, uint[] calldata fees) external onlyDistributorOwner
lock {
```

Action recommended: Remove unused variables.

Contract Size Limit

Herdius mentioned a problem consisting of large smart contract sizes. Currently, the smart contract size limit sits at 2**14 + 2**13 or 24576 bytes while the HerdiusMarket.sol contract has a size of 47582 bytes, HerdiusFactory.sol has 27996 bytes and HerdiusDeployProxy.sol which has a size of 48962 bytes. While there are discussions to remove or increase the current size limit (https://ethereum-magicians.org/t/removing-or-increasing-the-contract-size-limit/3045) and (https://github.com/ethereum/EIPs/issues/1662), no action has been taken so far.

Actions recommended:

- Shorten the require reason strings

Example:

```
  require(_penaltyFrom >= _amount, 'Amount is too small or not equal to 
the fees needed.');
```

Becomes

```
  require(_penaltyFrom >= _amount, 'Amount is not enough.');
```

- Remove Internal Functions

Removing internal functions and adding the code to the public function can help reduce the smart contract size.

```
function _burnCheck(uint hAmount, uint balance, uint penalty, uint 
liquidity) internal view returns (uint) {
        if (hAmount <= 0) return 0;
        if (balance - hAmount <= penalty && penalty > 0) return 0;
        if (liquidity < hAmount.add(penalty)) return 0;
        return 1;

    }
```

```
uint allowed = _burnCheck(hAmount, burnVars.accountBalance, 
burnVars._penaltyAmount, burnVars.liquidity);
        require(allowed == 1, 'Burn not allowed');
```

Becomes

```
uint allowed = 1;
if (hAmount <= 0) allowed=0;
if (burnVars.accountBalance - hAmount <= burnVars._penaltyAmount && 
burnVars._penaltyAmount > 0) allowed=0;
if (burnVars.liquidity < hAmount.add(burnVars._penaltyAmount)) allowed=0;
```

```
require(allowed == 1, 'Burn not allowed');
```

- Use Functions Instead Of Modifiers

Modifiers can be a factor in increasing the smart contract size. Using functions to check conditions will help reduce the contract size.

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or for damages resulting from the use of the provided information. Additionally Arcadia would like to emphasize that use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period of time.