Unlockd Finance - Debt Market

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 6th, 2023 - April 10th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/07/2023	Miguel Jalon
0.2	Document Edits	04/07/2023	Miguel Jalon
0.3	Document Edits	04/09/2023	Miguel Jalon
0.4	Draft Review	04/10/2023	Ataberk Yavuzer
0.5	Draft Review	04/11/2023	Piotr Cielas
0.6	Draft Review	04/12/2023	Gabi Urrutia
1.0	Remediation Plan	04/28/2023	Miguel Jalon
1.1	Remediation Plan Review	04/28/2023	Ataberk Yavuzer
1.2	Remediation Plan Review	04/28/2023	Piotr Cielas
1.3	Remediation Plan Review	05/02/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com
Miguel Jalon	Halborn	Miguel.Jalon@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Unlockd Finance is a Decentralized Finance protocol that allows users to lend, borrow, and earn interest on their NFTs. The Debt Market/Sell Loans feature allows borrowers to sell their debt.

Unlockd Finance engaged Halborn to conduct a security audit on their smart contracts beginning on March 6th, 2023 and ending on April 10th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided five weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Unlockd Finance team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the contracts' solidity code and can

quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing with custom scripts. (Foundry).
- Static Analysis of security for scoped contract, and imported functions manually.
- Testnet deployment (Anvil).

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. Unlockd

- Repository: Unlockd
- Commit ID: beb005be1073420e59403828a6597aaf0c698297
- Smart Contracts in scope:
    1. DebtMarket.sol
    2. DebtToken.sol
    3. IncentivizedERC20.sol
    4. InterestRate.sol
    5. LendPool.sol
    6. LendPoolAddressesProvider.sol
    7. LendPoolAddressesProviderRegistry.sol
    8. LendPoolConfigurator.sol
    9. LendPoolLoan.sol
    10. LendPoolStorage.sol
    11. LockeyManager.sol
    12. NFTOracle.sol
    13. ReserveOracle.sol
    14. UToken.sol
    15. WETHGateway.sol
    16. PunkGateway.sol
- Fixed Commit ID: 1785e82f2b7229c9f4d650cee5f3848a2b55482c

Out-of-scope:

1. Updates to the above contracts unrelated to the Debt Market/Sell Loans feature
2. Other contracts in the repository
3. Economic attacks
4. Dependencies and third-party packages

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	0	6

LIKELIHOOD

IMPACT

			(HAL-01)	
(HAL-02) (HAL-03) (HAL-04)				
(HAL-05) (HAL-06) (HAL-07)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) DENIAL OF SERVICE IN DEBT MARKET	High	SOLVED - 04/23/2023
(HAL-02) THE DEBTMARKET CONTRACT IS NOT PAUSABLE	Informational	SOLVED - 04/23/2023
(HAL-03) NO CAP FOR THE DELTABIDPERCENT PARAMETER	Informational	SOLVED - 04/23/2023
(HAL-04) CUSTOM NONREENTRANT MODIFIER USED	Informational	ACKNOWLEDGED
(HAL-05) SUBOPTIMAL ORDER OF OPERANDS IN LOGICAL EXPRESSIONS	Informational	SOLVED - 04/23/2023
(HAL-06) MISSING ZERO ADDRESS CHECK	Informational	ACKNOWLEDGED
(HAL-07) SOLC 0.8.4 COMPILER VERSION CONTAINS MULTIPLE BUGS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) DENIAL OF SERVICE IN DEBT MARKET - HIGH

Description:

With the cancelDebtListing function in the DebtMarket contracts, users can cancel their debt listings by providing the address of the relevant NFT and token ID.
Missing access control measures in the cancelDebtListing function in the DebtMarket contract allow any user to cancel any listing even if they are not the owner of those loans. This causes a denial of service of the main feature of the DebtMarket contract, as anyone can potentially cancel any listing.

Code Location:

Listing 1: DebtMarket.sol

```
193   function cancelDebtListing(
194     address nftAsset,
195     uint256 tokenId
196   ) external override nonReentrant debtShouldExistGuard(nftAsset,
 ↳ tokenId) {
197     uint256 debtId = _nftToDebtIds[nftAsset][tokenId];
198
199     DataTypes.DebtMarketListing storage sellDebt = _marketListings
 ↳ [debtId];
200     require(sellDebt.state != DataTypes.DebtMarketState.Sold,
 ↳ Errors.DM_DEBT_SHOULD_NOT_BE_SOLD);
201     sellDebt.state = DataTypes.DebtMarketState.Canceled;
202     _deleteDebtOfferListing(nftAsset, tokenId);
203
204     _nftToDebtIds[nftAsset][tokenId] = 0;
205
206     if (sellDebt.bidderAddress != address(0)) {
207       IERC20Upgradeable(sellDebt.reserveAsset).safeTransferFrom(
208         address(this),
209         sellDebt.bidderAddress,
210         sellDebt.bidPrice
211       );
```

```
212     }
213
214     emit DebtListingCanceled(
215       sellDebt.debtor,
216       sellDebt.debtId,
217       sellDebt,
218       _totalDebtsByCollection[nftAsset],
219       _userTotalDebtByCollection[sellDebt.debtor][nftAsset]
220     );
221   }
```

Proof Of Concept:

The following scenario describes the potential impact of this issue:

- Alice has a loan
- Alice cannot pay the loan
- Alice wants to sell the Loan in the debt market
- Alice creates a debt
- Bobby cancels Alice's debt
- Alice can't sell the loan and loses the collateral

Listing 2: DebtMarketTest.t.sol

```
1     function testCannotCancelOthersDebt() public {
2         // ALICE CREATES A DEBT LISTING
3         console.log("CONTEXT: ALICE HAS A LOAN THAT CAN NOT REPAY"
↳ );
4         console.log("ALICE PROCEEDS TO SELL THE LOAN TO THE DEBT
↳ MARKET");
5         vm.startPrank(alice);
6         address nftAsset = address(MFERS);
7         uint256 tokenId = 1;
8         uint256 sellPrice = 80;
9         address onBehalfOf = alice;
10        uint256 startBiddingPrice = 40;
11        uint256 auctionEndTimestamp = timeNow + 1 days;
12        debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
13        vm.stopPrank();
```

```
14
15          // BOBBY CANCELS DEBT LISTING
16          console.log("BOBBY TRIES TO CANCEL ALICES DEBT LISTING (
↳ REVERT EXPECTED)");
17          vm.prank(bobby);
18          vm.expectRevert();
19          debtMarket.cancelDebtListing(nftAsset, tokenId);
20      }
```

The transaction does not revert when Bobby is canceling Alice's debt listing:

```
Running 1 test for test/DebtMarketTest.t.sol:DebtMarketTest
[FAIL. Reason: Call did not revert as expected] testCannotCancelOthersDebt() (gas: 969228)
Logs:
  CONTEXT: ALICE HAS A LOAN THAT CAN NOT REPAY
  ALICE PROCEEDS TO SELL THE LOAN TO THE DEBT MARKET
  BOBBY TRIES TO CANCEL ALICES DEBT LISTING (REVERT EXPECTED)

Test result: FAILED. 0 passed; 1 failed; finished in 9.70ms

Failing tests:
Encountered 1 failing test in test/DebtMarketTest.t.sol:DebtMarketTest
[FAIL. Reason: Call did not revert as expected] testCannotCancelOthersDebt() (gas: 969228)
```

```
    |   |   ├─ [37969] MintableERC721::safeTransferFrom(0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c, LendPool: [0x642CE0f9
    |   |   |   ├─ emit Transfer(from: 0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c, to: LendPool: [0x642CE0f961d16178AeEd2
    |   |   |   ├─ [1022] LendPool::onERC721Received(LendPool: [0x642CE0f961d16178AeEd210d0d6dF71229149095], 0x61A1D7fD8C
    |   |   |   |   └─ ← 0x150b7a02
    |   |   |   └─ ← ()
    |   |   ├─ [23217] MintableERC721::approve(LendPoolLoan: [0xDB4a2FFD90ae982635d788Ec719947D5D75b3b79], 0)
    |   |   |   ├─ emit Approval(owner: LendPool: [0x642CE0f961d16178AeEd210d0d6dF71229149095], approved: LendPoolLoan: [
    |   |   |   └─ ← ()
    |   |   ├─ [162126] LendPoolLoan::createLoan(0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c, 0x61A1D7fD8C9bbd82932D99DFD4
 MintableERC20: [0xf13AE1dBa3E3f723D3dD95eB28660a55794EA9C9], 1, 1000000000000000000000000000)
    |   |   |   ├─ [28544] MintableERC721::safeTransferFrom(LendPool: [0x642CE0f961d16178AeEd210d0d6dF71229149095], LendP
    |   |   |   |   ├─ emit Transfer(from: LendPool: [0x642CE0f961d16178AeEd210d0d6dF71229149095], to: LendPoolLoan: [0xD
    |   |   |   |   ├─ [941] LendPoolLoan::onERC721Received(LendPoolLoan: [0xDB4a2FFD90ae982635d788Ec719947D5D75b3b79], L
    |   |   |   |   |   └─ ← 0x150b7a02
    |   |   |   |   └─ ← ()
    |   |   |   ├─ emit LoanCreated(user: 0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c, onBehalfOf: 0x61A1D7fD8C9bbd82932D9
veAsset: MintableERC20: [0xf13AE1dBa3E3f723D3dD95eB28660a55794EA9C9], amount: 1, borrowIndex: 1000000000000000000000000000
    |   |   |   └─ ← 0
    |   |   ├─ emit Borrow(user: 0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c, reserve: MintableERC20: [0xf13AE1dBa3E3f723D
 onBehalfOf: 0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c, borrowRate: 0, loanId: 0, referral: 0)
    |   |   └─ ← ()
    |   └─ ← ()
    ├─ [0] VM::stopPrank()
    |   └─ ← ()
    ├─ [0] console::log(CONTEXT: ALICE HAS A LOAN THAT CAN NOT REPAY) [staticcall]
    |   └─ ← ()
    ├─ [0] console::log(ALICE PROCEEDS TO SELL THE LOAN TO THE DEBT MARKET) [staticcall]
    |   └─ ← ()
    ├─ [0] VM::startPrank(0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c)
    |   └─ ← ()
    ├─ [328762] DebtMarket::createDebtListing(MintableERC721: [0x1D17d8E02947DC98921AC8aB19A98D6C5F27c119], 1, 80, 0x61A1
    |   ├─ [513] LendPoolAddressesProvider::getLendPoolLoan() [staticcall]
    |   |   └─ ← LendPoolLoan: [0xDB4a2FFD90ae982635d788Ec719947D5D75b3b79]
    |   ├─ [2726] LendPoolLoan::getCollateralLoanId(MintableERC721: [0x1D17d8E02947DC98921AC8aB19A98D6C5F27c119], 1) [sta
    |   |   └─ ← 0
    |   ├─ [978] LendPoolLoan::getLoanCollateralAndReserve(0) [staticcall]
    |   |   └─ ← MintableERC721: [0x1D17d8E02947DC98921AC8aB19A98D6C5F27c119], 0, MintableERC20: [0xf13AE1dBa3E3f723D3dD9
    |   ├─ emit DebtListingCreated(debtId: 1, debtor: 0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c, nftAsset: MintableERC72
20: [0xf13AE1dBa3E3f723D3dD95eB28660a55794EA9C9], debtAmount: 1, auctionEndTimestamp: 86401, startBiddingPrice: 40)
    |   └─ ← ()
    ├─ [0] VM::stopPrank()
    |   └─ ← ()
    ├─ [0] console::log(BOBBY TRIES TO CANCEL ALICES DEBT LISTING (REVERT EXPECTED)) [staticcall]
    |   └─ ← ()
    ├─ [0] VM::prank(0x3CE907fF40299087175b849632c8e4979C3ebABF)
    |   └─ ← ()
    ├─ [0] VM::expectRevert()
    |   └─ ← ()
    ├─ [30420] DebtMarket::cancelDebtListing(MintableERC721: [0x1D17d8E02947DC98921AC8aB19A98D6C5F27c119], 1)
    |   ├─ emit DebtListingCanceled(onBehalfOf: 0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c, debtId: 1, marketListing: (1,
95eB28660a55794EA9C9, 1, 0x0000000000000000000000000000000000000000000000000000, 0, 86401, 40), totalByCollection: 0, totalByUserAndC
    |   └─ ← ()
    └─ ← "Call did not revert as expected"

Test result: FAILED. 0 passed; 1 failed; finished in 9.76ms

Failing tests:
Encountered 1 failing test in test/DebtMarketTest.t.sol:DebtMarketTest
[FAIL. Reason: Call did not revert as expected] testCannotCancelOthersDebt() (gas: 969228)
```

Risk Level:

Likelihood - 4

Impact - 4

Recommendation:

Check that the only address allowed to execute cancelDebtListing is the owner of the debtListing.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue by adding a check if is the debtor of the NFT id or authorised address:

DebtMarket.sol - L222 // COMMIT ID: 1785e82f2b7229c9f4d650cee5f3848a2b55482c

3.2 (HAL-02) THE DEBTMARKET CONTRACT IS NOT PAUSABLE - INFORMATIONAL

Description:

When a contract is not pausable, if a bug is detected or an attack is executed, a lot of value could be lost for the protocol and the users stakeholders. It is recommended to add a pausable functionality to the contract to have a better control of the code and freeze the state if a problem is detected.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Adding the pause functionality to the DebtMarket contract is recommended: OpenZeppelin Pausable Security Contract

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue by adding a whenNotPaused modifier and a onlyPoolOwner function to pause the contract:

DebtMarket.sol - L518 // COMMIT ID: 1785e82f2b7229c9f4d650cee5f3848a2b55482c

3.3 (HAL-03) NO CAP FOR THE DELTABIDPERCENT PARAMETER - INFORMATIONAL

Description:

A high deltaBidPercent mistakenly introduced could disable bids and have an impact on the final debt sell price.

Code Location:

Listing 3: DebtMarket.sol

```
472   function setDeltaBidPercent(uint256 value) external override
 ↳ nonReentrant onlyPoolAdmin {
473     _deltaBidPercent = value;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to add a cap to the deltaBidPercent parameter to avoid setting a value that can disable a bid.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue by adding a cap to the value parameter for the setDeltaBidPercent function:

DebtMarket.sol - L499 // COMMIT ID: 1785e82f2b7229c9f4d650cee5f3848a2b55482c

3.4 (HAL-04) CUSTOM NONREENTRANT MODIFIER USED - INFORMATIONAL

Description:

In the DebtMarket contract, a custom nonReentrant modifier is used instead of using the already existing and security audited OpenZeppelin libraries for this purpose.

Code Location:

Listing 4: DebtMarket.sol

```
58   modifier nonReentrant() {
59     // On the first call to nonReentrant, _notEntered will be true
60     require(_status != _ENTERED, "ReentrancyGuard: reentrant call"
61     // Any calls to nonReentrant after this point will fail
62     _status = _ENTERED;
63     _;
64     // By storing the original value once again, a refund is
↳ triggered (see
65     // https://eips.ethereum.org/EIPS/eip-2200)
66     _status = _NOT_ENTERED;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

As a best practice, using OpenZeppelin libraries is recommended.

Remediation Plan:

ACKNOWLEDGED: The Unlockd Finance team acknowledged this issue.

3.5 (HAL-05) SUBOPTIMAL ORDER OF OPERANDS IN LOGICAL EXPRESSIONS - INFORMATIONAL

Description:

The order of logical values in a conjunction (AND) is an important factor to consider when optimizing gas usage in a contract. First, the left part of the expression is evaluated. In the case the first operand is false, the EVM does not keep evaluating the expression because its logical value is false if one of its operands is false. If the left operand is often true, generally the EVM needs to evaluate both operands to determine the result of the conjunction, increasing gas consumption.

Code Location:

Listing 5: DebtMarket.sol (Line 305)

```
293     require(
294       marketListing.sellType == DataTypes.DebtMarketType.Auction
 ↳ ||
295         marketListing.sellType == DataTypes.DebtMarketType.Mixed,
296       Errors.DM_INVALID_SELL_TYPE
297     );
298     require(bidPrice >= marketListing.startBiddingPrice, Errors.
 ↳ DM_BID_PRICE_LESS_THAN_MIN_BID_PRICE);
299     if (marketListing.sellType == DataTypes.DebtMarketType.Mixed)
 ↳ {
300       require(bidPrice <= vars.sellPrice, Errors.
 ↳ DM_BID_PRICE_HIGHER_THAN_SELL_PRICE);
301     }
302
303     require(block.timestamp <= marketListing.auctionEndTimestamp,
 ↳ Errors.DM_AUCTION_ALREADY_ENDED);
304
305     if (marketListing.sellType == DataTypes.DebtMarketType.Mixed
 ↳ && bidPrice == vars.sellPrice) {
306       // Sell the debt
307       _transferDebt(nftAsset, tokenId, onBehalfOf);
```

```
308         vars.price = vars.sellPrice;
309       } else {
310         require(
311           bidPrice > (marketListing.bidPrice + marketListing.
 ↳ bidPrice.percentMul(_deltaBidPercent)),
312           Errors.DM_BID_PRICE_LESS_THAN_PREVIOUS_BID
313         );
314         marketListing.state = DataTypes.DebtMarketState.Active;
315       }
316       marketListing.bidderAddress = onBehalfOf;
317       marketListing.bidPrice = vars.price;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

A more optimized check would be the one which checks the less likely condition first.

Listing 6: DebtMarket.sol (Line 305)

```
305       if (bidPrice == vars.sellPrice && marketListing.sellType ==
 ↳ DataTypes.DebtMarketType.Mixed) {
306         // Sell the debt
307         _transferDebt(nftAsset, tokenId, onBehalfOf);
308         vars.price = vars.sellPrice;
309       } else {
310         require(
311           bidPrice > (marketListing.bidPrice + marketListing.
 ↳ bidPrice.percentMul(_deltaBidPercent)),
312           Errors.DM_BID_PRICE_LESS_THAN_PREVIOUS_BID
313         );
314         marketListing.state = DataTypes.DebtMarketState.Active;
315       }
```

This optimization saves 120 gas:

```
    ├─ [84440] DebtMarket::bid(MintableERC721: [0x1D17d8E02947DC98921AC8aB19A98D6C5F27c119], 0, 40, 0x25418F13C03daaF4b37
    |   ├─ [22178] MintableERC20::transferFrom(0x25418F13C03daaF4b37751621cd96EE2566995dE, DebtMarket: [0xe6CCdc781deb32A
    |   |   ├─ emit Approval(owner: 0x25418F13C03daaF4b37751621cd96EE2566995dE, spender: DebtMarket: [0xe6CCdc781deb32A04
    |   |   ├─ emit Transfer(from: 0x25418F13C03daaF4b37751621cd96EE2566995dE, to: DebtMarket: [0xe6CCdc781deb32A0463fb1d
    |   |   └─ ← 0x0000000000000000000000000000000000000000000000000000000000000001
    |   ├─ emit BidPlaced(bidderAddress: 0x25418F13C03daaF4b37751621cd96EE2566995dE, reserveAsset: MintableERC20: [0xf13A
ebtId: 1, bidPrice: 40)
    |   └─ ← 0x0000000000000000000000000000000000000000000000000000000000000001
    ├─ [84560] DebtMarket::bid(MintableERC721: [0x1D17d8E02947DC98921AC8aB19A98D6C5F27c119], 0, 40, 0x25418F13C03daaF4b37
    |   ├─ [22178] MintableERC20::transferFrom(0x25418F13C03daaF4b37751621cd96EE2566995dE, DebtMarket: [0xe6CCdc781deb32A
    |   |   ├─ emit Approval(owner: 0x25418F13C03daaF4b37751621cd96EE2566995dE, spender: DebtMarket: [0xe6CCdc781deb32A04
    |   |   ├─ emit Transfer(from: 0x25418F13C03daaF4b37751621cd96EE2566995dE, to: DebtMarket: [0xe6CCdc781deb32A0463fb1d
    |   |   └─ ← 0x0000000000000000000000000000000000000000000000000000000000000001
    |   ├─ emit BidPlaced(bidderAddress: 0x25418F13C03daaF4b37751621cd96EE2566995dE, reserveAsset: MintableERC20: [0xf13A
ebtId: 1, bidPrice: 40)
    |   └─ ← ()
```

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue by inverting the order of the conditions to be checked:

DebtMarket.sol - L329 // COMMIT ID: 1785e82f2b7229c9f4d650cee5f3848a2b55482c

3.6 (HAL-06) MISSING ZERO ADDRESS CHECK - INFORMATIONAL

Description:

In the `initialize()` function, the parameter `_addressesProvider` is not checked to prevent pointing to the zero address. In case `_addressesProvider` is actually the zero address, the problem would be that it cannot catch other contracts to operate with, so the contract should be deployed again.

Code Location:

Listing 7: DirectLoanFixedCollectionOffer2.sol (Line 135)

```
134   function initialize(ILendPoolAddressesProvider addressesProvider
 ↳ ) external initializer {
135     _addressesProvider = addressesProvider;
136     _deltaBidPercent = PercentageMath.ONE_PERCENT;
137   }
138
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

When setting an address variable, always make sure the value is not zero.

Remediation Plan:

ACKNOWLEDGED: The Unlockd Finance team acknowledged this issue.

3.7 (HAL-07) SOLC 0.8.4 COMPILER VERSION CONTAINS MULTIPLE BUGS - INFORMATIONAL

Description:

The scoped contracts have configured the fixed pragma set to 0.8.4. The latest solidity compiler version, 0.8.19, fixed important bugs in the compiler along with new native protections. The current version is missing the following fixes: 0.8.5, 0.8.6, 0.8.7, 0.8.8, 0.8.9, 0.8.12, 0.8.13, 0.8.14, 0.8.15, 0.8.16, 0.8.17, 0.8.18, 0.8.19.

The official Solidity's recommendations are that you should use the latest released version of Solidity when deploying contracts. Apart from exceptional cases, only the newest version receives security fixes.

Code Location:

Listing 8: DirectLoanFixedCollectionOffer2.sol (Line 2)

```
1 // SPDX-License-Identifier: agpl-3.0
2 pragma solidity 0.8.4;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use the latest Solidity compiler version as possible.

Remediation Plan:

ACKNOWLEDGED: The Unlockd Finance team acknowledged this issue.

RETESTING

The issue described in this section was brought to Halborn's attention by the Unlockd Finance team during the engagement.

4.1 UNLOCKD01 - DEBT IS NOT RETIRED FROM LISTING WHEN LOAN IS REPAID

Description:

Please note, this finding was brought to Halborn's attention by the Unlockd Finance team.

When a loan is listed in the debt market, but then the loan is repaid, the debt is not properly canceled. The problem is that if the user wants to take another loan over the same NFT, as the debt listing is linked to the nft id and the nft address, anyone could buy or bid for the debt of the NFT without the borrower's willingness, getting then the NFT in question.

The repayment internal function in the Borrownig Logic is not tracking if there's a debt listing to cancel. Here is the function where the cancelation is missing:

Code Location:

Listing 9: BorrowLogic.sol

```
234   function _repay(
235     ILendPoolAddressesProvider addressesProvider,
236     mapping(address => DataTypes.ReserveData) storage reservesData
 ↳ ,
237     mapping(address => DataTypes.NftData) storage nftsData,
238     mapping(address => mapping(uint256 => DataTypes.
 ↳ NftConfigurationMap)) storage nftsConfig,
239     DataTypes.ExecuteRepayParams memory params
240   ) internal returns (uint256, bool) {
241     RepayLocalVars memory vars;
242     vars.initiator = params.initiator;
```

```
243
244     vars.poolLoan = addressesProvider.getLendPoolLoan();
245
246     vars.loanId = ILendPoolLoan(vars.poolLoan).getCollateralLoanId
 ↳ (params.nftAsset, params.nftTokenId);
247     require(vars.loanId != 0, Errors.
 ↳ LP_NFT_IS_NOT_USED_AS_COLLATERAL);
248
249     DataTypes.LoanData memory loanData = ILendPoolLoan(vars.
 ↳ poolLoan).getLoan(vars.loanId);
250
251     DataTypes.ReserveData storage reserveData = reservesData[
 ↳ loanData.reserveAsset];
252     DataTypes.NftData storage nftData = nftsData[loanData.nftAsset
 ↳ ];
253     DataTypes.NftConfigurationMap storage nftConfig = nftsConfig[
 ↳ params.nftAsset][params.nftTokenId];
254
255     // update state MUST BEFORE get borrow amount which is depent
 ↳ on latest borrow index
256     reserveData.updateState();
257
258     (, vars.borrowAmount) = ILendPoolLoan(vars.poolLoan).
 ↳ getLoanReserveBorrowAmount(vars.loanId);
259
260     ValidationLogic.validateRepay(reserveData, nftData, nftConfig,
 ↳  loanData, params.amount, vars.borrowAmount);
261
262     vars.repayAmount = vars.borrowAmount;
263     vars.isUpdate = false;
264     if (params.amount < vars.repayAmount) {
265       vars.isUpdate = true;
266       vars.repayAmount = params.amount;
267     }
268
269     if (vars.isUpdate) {
270       ILendPoolLoan(vars.poolLoan).updateLoan(
271         vars.initiator,
272         vars.loanId,
273         0,
274         vars.repayAmount,
275         reserveData.variableBorrowIndex
276       );
277     } else {
```

```
278       ILendPoolLoan(vars.poolLoan).repayLoan(
279         vars.initiator,
280         vars.loanId,
281         nftData.uNftAddress,
282         vars.repayAmount,
283         reserveData.variableBorrowIndex
```

Invariant Testing:

The particular case is forced in the handler contract to make the test case is proved.

Listing 10: Handler.sol

```
255     function borrowDebtListingAndRepay(uint256 amount, uint256
 ↳ nftTokenId, uint256 sellPrice, uint256 extraTime) public
 ↳ createActor() {
256         console.log("BORROW ", currentActor);
257 
258         sellPrice = bound(sellPrice, 1e16, 1e24);
259         extraTime = bound(extraTime, 0, 1000);
260 
261         vm.warp(block.timestamp + 1 days);
262         amount = bound(amount, 1e16, 1000e18);
263         nftTokenId = nftTokenId % 9999;
264 
265         try MFERS.mint(nftTokenId) {
266 
267             address asset = address(WETH);
268             address nftAsset = address(MFERS);
269 
270             MFERS.approve(address(lendPool), nftTokenId);
271             address onBehalfOf = currentActor;
272             lendPool.borrow(asset, amount, nftAsset, nftTokenId,
 ↳ onBehalfOf, 0);
273 
274             lastBorrowNftId = nftTokenId;
275             lastActor = currentActor;
276             loanCounter++;
277             toggleRandomnessCD = !toggleRandomnessCD;       // set
 ↳  to false
278 
```

```
279             uint256 startBiddingPrice = sellPrice/2;
280             uint256 auctionEndTimestamp = block.timestamp +
 ↳ extraTime * 1 days;
281
282             debtMarket.createDebtListing(nftAsset, nftTokenId,
 ↳ sellPrice, currentActor, startBiddingPrice, auctionEndTimestamp);
283
284             nftIdsWithDebt.push(lastBorrowNftId);
285
286             vm.warp(block.timestamp + 1 days);
287
288             lendPool.repay(nftAsset, nftTokenId, amount);
289         }
290         catch {}
291     }
```

The property that is being checked is that for every debt listed in the debt market, the NFT linked to the loan corresponding to the given debt, has always to be under protocol's control.

Listing 11: DebtInvariantTesting.sol

```
332     function invariant_DebtMarket() public {
333
334         console.log("
 ↳ ------------------------------------------------------");
335         console.log("INVARIANT: FOR EVERY DEBT LISTED IN THE
 ↳ CONTRACT,");
336         console.log("THE NFT LINKED TO ITS CORRESPONDING LOAN HAS
 ↳ TO BE UNDER THE PROTOCOL'S POSSESION");
337         console.log("
 ↳ ------------------------------------------------------");
338
339         uint256[] memory debts = handler.getnftIdsWithDebt();
340         uint256 length = debts.length;
341
342         console.log("GETING AN ARRAY OF THE LOANS LISTED IN THE
 ↳ DEBT MARKET...      ");
343         console.log("CHECKING EVERY NFT LISTED IS STILL WITHIN THE
 ↳  PROTOCOL...      ");
344
345         // uint256 currentDebt = countTemp.current();
346         console.log("CHECKING CURRENT DEBT --> ", length);
```

```
347         console.log("
 ↳ ------------------------------------------------------------");
348
349         for (uint i; i < length; i++){
350             assertEq(MFERS.ownerOf(debts[i]), address(finalProxy))
 ↳ ;
351             console.log("OWNER OF NFT:    ", MFERS.ownerOf(debts[i
 ↳ ]));
352             console.log("UNFT PROXY:      ", address(finalProxy));
353         }
354
355         console.log("
 ↳ ------------------------------------------------------------");
356     }
```

As we can see here, when a repayment is executed, the NFT is in a different address (the user's one) from the proxy used for this particular NFT Collection.

```
Running 1 test for test/DebtInvariantTest.t.sol:DebtInvariantTest
[FAIL. Reason: Assertion failed.]
        [Sequence]
                sender=0x775984a1337d15dc1bf4ce50f41988b2a2b34ffb addr=[test/Handler.sol:Handler]0xa46edc8aec511f1fa501b40ffd74b1
201979907828, 24057271466746910189010981540035688271099661731674933891684532706201979912908, 563545113218786458200636276444874205

 invariant_DebtMarket() (runs: 1, calls: 1, reverts: 0)
Logs:
  BORROW  0x775984a1337d15Dc1bf4Ce50f41988b2a2B34ffb
  Bound Result 283372772870982457115884
  Bound Result 152
  Bound Result 999990000000000005651
  ------------------------------------------------------------------------------------------------
  INVARIANT: FOR EVERY DEBT LISTED IN THE CONTRACT,
  THE NFT LINKED TO ITS CORRESPONDING LOAN HAS TO BE UNDER THE PROTOCOL'S POSSESION
  ------------------------------------------------------------------------------------------------
  GETING AN ARRAY OF THE LOANS LISTED IN THE DEBT MARKET...
  CHECKING EVERY NFT LISTED IS STILL WITHIN THE PROTOCOL...
  CHECKING CURRENT DEBT -->  1
  ------------------------------------------------------------------------------------------------
  Error: a == b not satisfied [address]
        Left: 0x01cb6c0007a0d23cc00c5a448ce4452502b73ac9
       Right: 0xe27f8c7903359d221f296fa9028ac7dba0c80062
  OWNER OF NFT:     0x01cb6c0007A0d23cC00c5A448Ce4452502b73aC9
  UNFT PROXY:       0xE27F8c7903359d221F296Fa9028aC7Dba0c80062
  ------------------------------------------------------------------------------------------------

Test result: FAILED. 0 passed; 1 failed; finished in 300.09ms
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to track if the loan over this NFT collateralized has a debt listed in the debt market. In this case, the listing should be cancelled.

Remediation Plan:

SOLVED: The Unlockd Finance team found this issue and solved it by cancelling the debt listing in case that it exists a debt listed for the loan that is being repaid:

BorrowLogic.sol - L286 // COMMIT ID: 1785e82f2b7229c9f4d650cee5f3848a2b55482c

RETESTING

MANUAL TESTING

5.1 TESTS METHODOLOGY

The tests for the DebtMarket audit were focused on the DebtMarket contract and the contracts directly affected by the introduction of this feature. Here's a list of the kinds of tests that were performed.

- Tests focused on bids
- Tests focused on buys
- Tests focused on claims
- Tests checking the functionality of WETHGateway
- Tests checking the functionality of the Punks

5.2 CODE

Listing 12: DebtMarketTest.t.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "forge-std/Test.sol";
// import "test/TestHelper.sol";
// import { ILSSVMPair } from "../../types/ILSSVMPair";
import "../src/mock/NFTX/NFTXVaultFactoryV2.sol";
// import { IUniswapV2Router02 } from "../../types/
↳ IUniswapV2Router02";
import { LendPool } from "../src/protocol/LendPool.sol";
import "../src/protocol/LendPoolAddressesProvider.sol";
import "../src/protocol/WETHGateway.sol";
import "../src/protocol/PunkGateway.sol";
import {ILendPoolAddressesProvider} from "../src/interfaces/
↳ ILendPoolAddressesProvider.sol";
import { LendPoolConfigurator } from "../src/protocol/
↳ LendPoolConfigurator.sol";
import "../src/mock/MintableERC20.sol";
import { LendPoolConfigurator } from "../src/protocol/
↳ LendPoolConfigurator.sol";
import "../src/mock/WETH9Mocked.sol";
import "../src/mock/MintableERC721.sol";
import "../src/mock/WrappedPunk/WrappedPunk.sol";
```

```
20 import "../src/mock/WrappedPunk/ICryptoPunk.sol";
21 import "../src/mock/CryptoPunksMarket.sol";
22 import "../src/mock/MintableERC721.sol";
23 // import { MockChainlinkOracle } from "../src/mock/
↳ MockChainlinkOracle";
24 // import { MockNFTOracle } from "../src/mock/MockNFTOracle";
25 import { MockReserveOracle } from "../src/mock/MockReserveOracle.
↳ sol";
26 import "../src/protocol/NFTOracle.sol";
27 import "../src/protocol/LendPoolLoan.sol";
28 import "../src/protocol/UToken.sol";
29 import "../src/protocol/DebtToken.sol";
30 import "../src/protocol/InterestRate.sol";
31 import "../src/protocol/LockeyManager.sol";
32 import "../src/mock/MockIncentivesController.sol";
33 import "../src/interfaces/INFTOracleGetter.sol";
34
35 //import "../src/mock/SushiSwap/uniswapv2/UniswapV2Router02.sol";
36 // import { ReserveOracle } from "../src/mock/ReserveOracle";
37 import "../src/mock/UNFT/UNFTRegistry.sol";
38 // import { UNFT } from "../src/mock/UNFT/UNFT";
39 import "../src/misc/UnlockdProtocolDataProvider.sol";
40 import "../src/misc/UiPoolDataProvider.sol";
41 import "../src/misc/WalletBalanceProvider.sol";
42 // import { UToken } from "../src/protocol/UToken";
43 import "../src/protocol/DebtMarket.sol";
44 import "../src/protocol/DebtToken.sol";
45
46 // import "../src/libraries/MyMath.sol";
47 // import "../src/libraries/LibBytes.sol";
48 // import "oz/token/ERC20/utils/SafeERC20.sol";
49
50 contract DebtMarketTest is Test {
51
52     // using MyMath for uint256;
53
54     address internal owner;
55     address internal treasury;
56     //address internal liqPr;
57     //address internal floan;
58     address internal alice;
59     address internal bobby;
60     address internal carla;
61     address internal edgar;
```

```
62      address internal zeroo;
63      DebtMarket internal debtMarket;
64      LendPool internal lendPool;
65      MockReserveOracle internal reserveOracle;
66      NFTOracle internal nftOracle;
67      WETHGateway internal wethGateway;
68      PunkGateway internal punkGateway;
69      LendPoolLoan internal lendPoolLoan;
70      UNFTRegistry internal unftRegistry;
71      MockIncentivesController internal incentivesController;
72      UnlockdProtocolDataProvider internal
↳ unlockdProtocolDataProvider;
73      UiPoolDataProvider internal uiDataProvider;
74      WalletBalanceProvider internal walletBalanceProvider;
75      NFTXVaultFactoryV2 internal nftxVaultFactoryV2;
76      LendPoolConfigurator internal lendPoolConfigurator;
77      LendPoolAddressesProvider internal lendPoolAddressesProvider;
78      WETH9Mocked internal WETH;
79      MintableERC20 internal USDT;
80      MintableERC721 internal BAYC;
81      MintableERC721 internal MFERS;
82      CryptoPunksMarket internal PUNK;
83      WrappedPunk internal wPUNK;
84      UToken internal uWETH;
85      UToken internal uUSDT;
86      DebtToken internal dWETH;
87      DebtToken internal dUSDT;
88
89      InterestRate internal interestRate;
90      uint256 internal timeNow = block.timestamp;
91
92      function setUp() public {
93          owner = vm.addr(0x60DDD);
94          treasury = vm.addr(0x11);
95          //liqPr = vm.addr(0x7181D);
96          //floan = vm.addr(0xF7A28);
97          alice = vm.addr(0xA71CE);
98          bobby = vm.addr(0xB0BB1);
99          carla = vm.addr(0xCA47A);
100         edgar = vm.addr(0xED6A4);
101         zeroo = address(0);
102
103         vm.deal(owner, 10000 ether);
104         vm.deal(alice, 1000 ether);
```

```
105         vm.deal(bobby, 1000 ether);
106         vm.deal(carla, 1000 ether);
107         vm.deal(edgar, 1000 ether);
108 
109         // DEPLOYING MOCKS
110         BAYC = new MintableERC721("Bored Ape Yatch Club", "BAYC");
111         MFERS = new MintableERC721("MFERS", "MFERS");
112         PUNK = new CryptoPunksMarket();
113         WETH = new WETH9Mocked();
114         USDT = new MintableERC20("USD Tether", "USDT", 6);
115         wPUNK = new WrappedPunk(address(PUNK));
116 
117         PUNK.allInitialOwnersAssigned();
118 
119         // MINTING NFTs (MFERS + PUNKS)
120         vm.prank(alice);
121         MFERS.mint(0);
122         vm.prank(carla);
123         MFERS.mint(1);
124         vm.prank(alice);
125         PUNK.getPunk(0);
126 
127         // DEPLOYING AND MINTING TOKEN
128         //vm.prank(alice);
129         vm.startPrank(owner);
130         WETH.mint(400_000000000000000000);
131         WETH.transfer(alice, 100_000000000000000000);
132         WETH.transfer(bobby, 100_000000000000000000);
133         WETH.transfer(carla, 100_000000000000000000);
134         WETH.transfer(edgar, 100_000000000000000000);
135         USDT.mint(4000_000000);
136         USDT.transfer(alice, 1000_000000);
137         USDT.transfer(bobby, 1000_000000);
138         USDT.transfer(carla, 1000_000000);
139         USDT.transfer(edgar, 1000_000000);
140 
141         // DEPLOY LEND POOL ADDRESSES PROVIDER
142         lendPoolAddressesProvider = new LendPoolAddressesProvider(
 ↳ "1");
143         lendPoolAddressesProvider.setAddress(bytes32("
 ↳ LEND_POOL_CONFIGURATOR"), owner);
144 
145         // INIT LEND POOL
146         debtMarket = new DebtMarket();
```

```
147         debtMarket.initialize(ILendPoolAddressesProvider(
 ↳ lendPoolAddressesProvider));
148 
149         // DEPLOY AND INIT UTOKEN AND DEBT TOKENS FOR WETH AND
 ↳ USDT
150         uWETH = new UToken();
151         uWETH.initialize(lendPoolAddressesProvider, treasury,
 ↳ address(WETH), 18, "uWETH", "uWETH" );
152         uUSDT = new UToken();
153         uUSDT.initialize(lendPoolAddressesProvider, treasury,
 ↳ address(USDT), 6, "uUSDT", "uUSDT" );
154         dWETH = new DebtToken();
155         dWETH.initialize(lendPoolAddressesProvider, address(WETH),
 ↳  18, "dWETH", "dWETH" );
156         dUSDT = new DebtToken();
157         dUSDT.initialize(lendPoolAddressesProvider, address(USDT),
 ↳  6, "dUSDT", "dUSDT" );
158 
159         // INTEREST RATE
160         interestRate = new InterestRate(lendPoolAddressesProvider,
 ↳  10, 10, 1, 2);
161 
162         // DEPLOY POOL
163         lendPool = new LendPool();
164 
165         // INIT POOLS
166         lendPool.initialize(ILendPoolAddressesProvider(
 ↳ lendPoolAddressesProvider));
167         lendPool.initReserve(address(USDT), address(uUSDT),
 ↳ address(dUSDT), address(interestRate));
168         lendPool.setReserveConfiguration(address(USDT),
 ↳ 80000000000000000);
169 
170         lendPool.initReserve(address(WETH), address(uWETH),
 ↳ address(dWETH), address(interestRate));
171         lendPool.setReserveConfiguration(address(WETH),
 ↳ 80000000000000000);
172 
173         //lendPoolAddressesProvider.setAddress(bytes32("LEND_POOL
 ↳ "), address(lendPool));
174 
175         wethGateway = new WETHGateway();
176         wethGateway.initialize(address(lendPoolAddressesProvider),
 ↳  address(WETH));
```

```
177
178         punkGateway = new PunkGateway();
179         punkGateway.initialize(address(lendPoolAddressesProvider),
 ↳  address(wethGateway), address(PUNK), address(wPUNK));
180
181         // "LEND_POOL_CONFIGURATOR";
182         lendPoolConfigurator = new LendPoolConfigurator();
183         lendPoolConfigurator.initialize(ILendPoolAddressesProvider
 ↳ (lendPoolAddressesProvider));
184
185         // "POOL_ADMIN";
186         lendPoolAddressesProvider.setPoolAdmin(owner);
187
188         // "EMERGENCY_ADMIN";
189         lendPoolAddressesProvider.setEmergencyAdmin(owner);
190
191         // "LEND_POOL_LIQUIDATOR";
192         lendPoolAddressesProvider.setLendPoolLiquidator(bobby);
193
194         // INITIALIZE LENDPOOL
195         bytes memory bytess;
196         lendPoolAddressesProvider.setLendPoolImpl(address(lendPool
 ↳ ), bytess);
197
198         // "RESERVE_ORACLE";
199         reserveOracle = new MockReserveOracle();
200
201         // "NFT_ORACLE";
202         nftOracle = new NFTOracle();
203
204         // "LEND_POOL_LOAN";
205         lendPoolLoan = new LendPoolLoan();
206
207         // "UNFT_REGISTRY";
208         unftRegistry = new UNFTRegistry();
209
210         // "INCENTIVES_CONTROLLER";
211         incentivesController = new MockIncentivesController();
212
213         // "UNLOCKD_DATA_PROVIDER";
214         unlockdProtocolDataProvider = new
 ↳ UnlockdProtocolDataProvider(ILendPoolAddressesProvider(
 ↳ lendPoolAddressesProvider));
215
```

```
216         // "UI_DATA_PROVIDER";
217         uiDataProvider = new UiPoolDataProvider(reserveOracle,
 ↳ INFTOracleGetter(address(nftOracle)));
218 
219         // "WALLET_BALANCE_PROVIDER";
220         walletBalanceProvider = new WalletBalanceProvider();
221 
222         // "NFTX_VAULT_FACTORY";
223         nftxVaultFactoryV2 = new NFTXVaultFactoryV2(address(owner)
 ↳ );
224 
225         // SETTING ADDRESSES
226         lendPoolAddressesProvider.setAddress(bytes32("LEND_POOL"),
 ↳  address(lendPool));
227         lendPoolAddressesProvider.setAddress(bytes32("
 ↳ RESERVE_ORACLE"), address(reserveOracle));
228         lendPoolAddressesProvider.setAddress(bytes32("NFT_ORACLE")
 ↳ , address(nftOracle));
229         lendPoolAddressesProvider.setAddress(bytes32("
 ↳ LEND_POOL_LOAN"), address(lendPoolLoan));
230         lendPoolAddressesProvider.setAddress(bytes32("DEBT_MARKET"
 ↳ ), address(debtMarket));
231 
232         vm.stopPrank();
233     }
234 
235     function customEnv1() internal {
236         vm.startPrank(edgar);
237         USDT.approve(address(lendPool), 100);
238         address asset = address(USDT);
239         uint256 amount = 100;
240         uint16 referralCode = 0;
241         lendPool.deposit(asset, amount, address(edgar),
 ↳ referralCode);
242         vm.stopPrank();
243 
244         vm.startPrank(alice);
245 
246         amount = 1;
247         address nftAsset = address(MFERS);
248         uint256 nftTokenId = 0;
249 
250         MFERS.approve(address(lendPool), 0);
251         lendPool.borrow(asset, amount, nftAsset, nftTokenId,
```

```
 ↳ address(alice), uint16(0));
252         vm.stopPrank();
253     }
254
255     function customEnv2() internal {
256         vm.startPrank(edgar);
257         address asset = address(USDT);
258         uint16 referralCode = 0;
259
260         address(wethGateway).call{value: 100}(
261             abi.encodeWithSignature("depositETH(address,uint16)",
 ↳ edgar, 0)
262         );
263         vm.stopPrank();
264
265         vm.startPrank(alice);
266         uint256 amount = 1;
267         address nftAsset = address(MFERS);
268         uint256 nftTokenId = 0;
269
270         MFERS.approve(address(lendPool), 0);
271
272         wethGateway.borrowETH(amount, nftAsset, nftTokenId,
 ↳ address(alice), uint16(0));
273         vm.stopPrank();
274     }
275
276
277     function testWETHBid() public {
278         customEnv2();
279
280         address nftAsset = address(MFERS);
281         uint256 tokenId = 0;
282         uint256 sellPrice = 80;
283         address onBehalfOf = alice;
284         uint256 startBiddingPrice = 40;
285         uint256 auctionEndTimestamp = timeNow + 1 days;
286         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
287         vm.stopPrank();
288
289         vm.startPrank(edgar);
290
291         address(wethGateway).call{value: 40}(
```

```
292             abi.encodeWithSignature("bidDebtETH(address,uint256,
 ↳ address)",  nftAsset, tokenId, edgar)
293         );
294     }
295 
296     function testWETHBidExceed() public {
297         customEnv2();
298 
299         address nftAsset = address(MFERS);
300         uint256 tokenId = 0;
301         uint256 sellPrice = 80;
302         address onBehalfOf = alice;
303         uint256 startBiddingPrice = 40;
304         uint256 auctionEndTimestamp = timeNow + 1 days;
305         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
306         vm.stopPrank();
307 
308         vm.startPrank(edgar);
309 
310         address(wethGateway).call{value: 90}(
311             abi.encodeWithSignature("bidDebtETH(address,uint256,
 ↳ address)",  nftAsset, tokenId, edgar)
312         );
313     }
314 
315     function testWETHBuy() public {
316         customEnv1();
317         address nftAsset = address(MFERS);
318         uint256 tokenId = 0;
319         uint256 sellPrice = 80;
320         address onBehalfOf = alice;
321         uint256 startBiddingPrice = 40;
322         uint256 auctionEndTimestamp = timeNow + 1 days;
323         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
324         vm.stopPrank();
325 
326         vm.startPrank(edgar);
327 
328         debtMarket.buy(nftAsset, tokenId, edgar, sellPrice);
329         address(wethGateway).call{value: 90}(
330             abi.encodeWithSignature("buyDebtETH(address,uint256,
 ↳ address)",  nftAsset, tokenId, edgar)
```

```
331         );
332     }
333 
334     function testPunks1() public {
335         customEnv1();
336 
337         address nftAsset = address(PUNK);
338         uint256 punkIndex = 0;
339         uint256 sellPrice = 80;
340         address onBehalfOf = alice;
341         uint256 startBiddingPrice = 40;
342         uint256 auctionEndTimestamp = timeNow + 1 days;
343 
344         debtMarket.createDebtListing(nftAsset, punkIndex,
 ↳ sellPrice, onBehalfOf, startBiddingPrice, auctionEndTimestamp);
345         vm.stopPrank();
346 
347         vm.startPrank(edgar);
348         USDT.approve(address(debtMarket), 40);
349 
350         address(wethGateway).call{value: 90}(
351             abi.encodeWithSignature("buyDebtPunk(uint256,address,
 ↳ uint256)",  nftAsset, punkIndex, edgar)
352         );
353     }
354 
355     function testPunks2() public {
356         customEnv1();
357 
358         address nftAsset = address(PUNK);
359         uint256 punkIndex = 0;
360         uint256 sellPrice = 80;
361         address onBehalfOf = alice;
362         uint256 startBiddingPrice = 40;
363         uint256 auctionEndTimestamp = timeNow + 1 days;
364 
365         debtMarket.createDebtListing(nftAsset, punkIndex,
 ↳ sellPrice, onBehalfOf, startBiddingPrice, auctionEndTimestamp);
366         vm.stopPrank();
367 
368         vm.startPrank(edgar);
369         USDT.approve(address(debtMarket), 40);
370         debtMarket.bid(nftAsset, punkIndex, startBiddingPrice,
 ↳ edgar);
```

```
371
372         vm.warp(8400000);
373         debtMarket.claim(nftAsset, punkIndex, edgar);
374     }
375
376     // function testFool() public {}
377
378     function testBid() public {
379         customEnv1();
380
381         address nftAsset = address(MFERS);
382         uint256 tokenId = 0;
383         uint256 sellPrice = 80;
384         address onBehalfOf = alice;
385         uint256 startBiddingPrice = 40;
386         uint256 auctionEndTimestamp = timeNow + 1 days;
387         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
388         vm.stopPrank();
389
390         vm.startPrank(edgar);
391         USDT.approve(address(debtMarket), 40);
392         debtMarket.bid(nftAsset, tokenId, startBiddingPrice, edgar
 ↳ );
393     }
394
395     function testBidExceed() public {
396         customEnv1();
397         address nftAsset = address(MFERS);
398         uint256 tokenId = 0;
399         uint256 sellPrice = 80;
400         address onBehalfOf = alice;
401         uint256 startBiddingPrice = 40;
402         uint256 auctionEndTimestamp = timeNow + 1 days;
403         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
404         vm.stopPrank();
405
406         vm.startPrank(edgar);
407         USDT.approve(address(debtMarket), 90);
408         debtMarket.bid(nftAsset, tokenId, 90, edgar);
409     }
410
411     function testCannotBidLess() public {
```

```
412         customEnv1();
413         address nftAsset = address(MFERS);
414         uint256 tokenId = 0;
415         uint256 sellPrice = 80;
416         address onBehalfOf = alice;
417         uint256 startBiddingPrice = 40;
418         uint256 auctionEndTimestamp = timeNow + 1 days;
419         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
420         vm.stopPrank();
421
422         vm.startPrank(edgar);
423         USDT.approve(address(debtMarket), 30);
424         vm.expectRevert();
425         debtMarket.bid(nftAsset, tokenId, 30, edgar);
426     }
427
428     function testBuy() public {
429         customEnv1();
430         address nftAsset = address(MFERS);
431         uint256 tokenId = 0;
432         uint256 sellPrice = 80;
433         address onBehalfOf = alice;
434         uint256 startBiddingPrice = 40;
435         uint256 auctionEndTimestamp = timeNow + 1 days;
436         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
437         vm.stopPrank();
438
439         vm.startPrank(edgar);
440         USDT.approve(address(debtMarket), 90);
441         debtMarket.buy(nftAsset, tokenId, edgar, sellPrice);
442     }
443
444     function testCannotBuyLess() public {
445         customEnv1();
446         address nftAsset = address(MFERS);
447         uint256 tokenId = 0;
448         uint256 sellPrice = 80;
449         address onBehalfOf = alice;
450         uint256 startBiddingPrice = 40;
451         uint256 auctionEndTimestamp = timeNow + 1 days;
452         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
```

```
453         vm.stopPrank();
454
455         vm.startPrank(edgar);
456         USDT.approve(address(debtMarket), 79);
457         vm.expectRevert();
458         debtMarket.buy(nftAsset, tokenId, edgar, 79);
459     }
460
461     function testCannotBuyMore() public {
462         customEnv1();
463         address nftAsset = address(MFERS);
464         uint256 tokenId = 0;
465         uint256 sellPrice = 80;
466         address onBehalfOf = alice;
467         uint256 startBiddingPrice = 40;
468         uint256 auctionEndTimestamp = timeNow + 1 days;
469         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
470         vm.stopPrank();
471
472         vm.startPrank(edgar);
473         USDT.approve(address(debtMarket), 81);
474         vm.expectRevert();
475         debtMarket.buy(nftAsset, tokenId, edgar, 81);
476     }
477
478     function testCannotClaim2Times() public {
479         customEnv1();
480         address nftAsset = address(MFERS);
481         uint256 tokenId = 0;
482         uint256 sellPrice = 80;
483         address onBehalfOf = alice;
484         uint256 startBiddingPrice = 40;
485         uint256 auctionEndTimestamp = timeNow + 1 days;
486         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
487         vm.stopPrank();
488
489         vm.startPrank(edgar);
490         USDT.approve(address(debtMarket), 81);
491
492         debtMarket.bid(nftAsset, tokenId, 50, edgar);
493
494         vm.warp(8400000);
```

```
495         debtMarket.claim(nftAsset, tokenId, edgar);
496         vm.expectRevert();
497         debtMarket.claim(nftAsset, tokenId, edgar);
498     }
499 
500     function testCannotCancelOthersDebt() public {
501         customEnv1();
502         // ALICE CREATES A DEBT LISTING
503         console.log("CONTEXT: ALICE HAS A LOAN THAT CAN NOT REPAY"
 ↳ );
504         console.log("ALICE PROCEEDS TO SELL THE LOAN TO THE DEBT
 ↳ MARKET");
505         vm.startPrank(alice);
506         address nftAsset = address(MFERS);
507         uint256 tokenId = 1;
508         uint256 sellPrice = 80;
509         address onBehalfOf = alice;
510         uint256 startBiddingPrice = 40;
511         uint256 auctionEndTimestamp = timeNow + 1 days;
512         debtMarket.createDebtListing(nftAsset, tokenId, sellPrice,
 ↳  onBehalfOf, startBiddingPrice, auctionEndTimestamp);
513         vm.stopPrank();
514 
515         // BOBBY CANCELS DEBT LISTING
516         console.log("BOBBY TRIES TO CANCEL ALICES DEBT LISTING (
 ↳ REVERT EXPECTED)");
517         vm.prank(bobby);
518         vm.expectRevert();
519         debtMarket.cancelDebtListing(nftAsset, tokenId);
520     }
```

AUTOMATED TESTING

6.1 STATIC ANALYSIS

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

```
MathUtils.calculateCompoundedInterest(uint256,uint40,uint256) (contracts/libraries/math/MathUtils.sol#39-64) performs a multiplicati
        - ratePerSecond = rate / SECONDS_PER_YEAR (contracts/libraries/math/MathUtils.sol#55)
        - WadRayMath.ray() + (ratePerSecond * (exp)) + (secondTerm) + (thirdTerm) (contracts/libraries/math/MathUtils.sol#63)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

MathUtils.calculateCompoundedInterest(uint256,uint40,uint256) (contracts/libraries/math/MathUtils.sol#39-64) uses a dangerous strict
        - exp == 0 (contracts/libraries/math/MathUtils.sol#47)
ReserveLogic.getNormalizedDebt(DataTypes.ReserveData) (contracts/libraries/logic/ReserveLogic.sol#76-90) uses a dangerous strict equ
        - timestamp == uint40(block.timestamp) (contracts/libraries/logic/ReserveLogic.sol#80)
ReserveLogic.getNormalizedIncome(DataTypes.ReserveData) (contracts/libraries/logic/ReserveLogic.sol#53-67) uses a dangerous strict e
        - timestamp == uint40(block.timestamp) (contracts/libraries/logic/ReserveLogic.sol#57)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in DebtMarket.bid(address,uint256,uint256,address) (contracts/protocol/DebtMarket.sol#277-333):
        External calls:
        - _transferDebt(nftAsset,tokenId,onBehalfOf) (contracts/protocol/DebtMarket.sol#307)
                - ILendPool(vars.lendPoolAddress).updateReserveState(loanData.reserveAsset) (contracts/protocol/DebtMarket.sol#413)
                - ILendPool(vars.lendPoolAddress).updateReserveInterestRates(loanData.reserveAsset) (contracts/protocol/DebtMarket.s
                - IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.borrowAmount,reserveData.variableBorrowIndex)
                - IDebtToken(reserveData.debtTokenAddress).mint(vars.buyer,vars.buyer,vars.borrowAmount,reserveData.variableBorrowIn
                - ILendPoolLoan(vars.lendPoolLoanAddress).reMintUNFT(loanData.nftAsset,loanData.nftTokenId,loanData.borrower,vars.bu
        State variables written after the call(s):
        - marketListing.bidderAddress = onBehalfOf (contracts/protocol/DebtMarket.sol#316)
        DebtMarket._marketListings (contracts/protocol/DebtMarket.sol#41) can be used in cross function reentrancies:
        - DebtMarket.getDebt(uint256) (contracts/protocol/DebtMarket.sol#486-488)
        - marketListing.bidPrice = vars.price (contracts/protocol/DebtMarket.sol#317)
        DebtMarket._marketListings (contracts/protocol/DebtMarket.sol#41) can be used in cross function reentrancies:
        - DebtMarket.getDebt(uint256) (contracts/protocol/DebtMarket.sol#486-488)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

DebtMarket.buy(address,uint256,address,uint256).vars (contracts/protocol/DebtMarket.sol#232) is a local variable never initialized
ReserveLogic.updateInterestRates(DataTypes.ReserveData,address,address,uint256,uint256).vars (contracts/libraries/logic/ReserveLogic
GenericLogic.calculateLoanLiquidatePrice(uint256,address,DataTypes.ReserveData,address,uint256,DataTypes.NftConfigurationMap,address
DebtMarket._transferDebt(address,uint256,address).vars (contracts/protocol/DebtMarket.sol#404) is a local variable never initialized
DebtMarket.claim(address,uint256,address).vars (contracts/protocol/DebtMarket.sol#343) is a local variable never initialized
GenericLogic.calculateLoanBidFine(address,DataTypes.ReserveData,address,DataTypes.NftConfigurationMap,DataTypes.LoanData,address,add
ReserveLogic._mintToTreasury(DataTypes.ReserveData,uint256,uint256,uint256,uint256,uint40).vars (contracts/libraries/logic/ReserveLo
GenericLogic.calculateNftDebtData(address,DataTypes.ReserveData,address,uint256,address).vars (contracts/libraries/logic/GenericLogi
DebtMarket.bid(address,uint256,uint256,address).vars (contracts/protocol/DebtMarket.sol#283) is a local variable never initialized
GenericLogic.calculateLoanData(address,DataTypes.ReserveData,address,uint256,DataTypes.NftConfigurationMap,address,uint256,address,a
DebtMarket.createDebtListing(address,uint256,uint256,address,uint256,uint256).vars (contracts/protocol/DebtMarket.sol#150) is a loca
GenericLogic.calculateOptimalMinRedeemValue(uint256,address,uint256,address,uint256,uint256).vars (contracts/libraries/logic/Generic
GenericLogic.calculateNftCollateralData(address,DataTypes.ReserveData,address,uint256,address,address).vars (contracts/libraries/log
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

DebtMarket._transferDebt(address,uint256,address) (contracts/protocol/DebtMarket.sol#403-450) ignores return value by IDebtToken(res
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

DebtMarket.setDeltaBidPercent(uint256) (contracts/protocol/DebtMarket.sol#472-474) should emit an event for:
        - _deltaBidPercent = value (contracts/protocol/DebtMarket.sol#473)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Reentrancy in DebtMarket._transferDebt(address,uint256,address) (contracts/protocol/DebtMarket.sol#403-450):
        External calls:
        - ILendPool(vars.lendPoolAddress).updateReserveState(loanData.reserveAsset) (contracts/protocol/DebtMarket.sol#413)
        - ILendPool(vars.lendPoolAddress).updateReserveInterestRates(loanData.reserveAsset) (contracts/protocol/DebtMarket.sol#414)
        State variables written after the call(s):
        - marketOrder.state = DataTypes.DebtMarketState.Sold (contracts/protocol/DebtMarket.sol#423)
Reentrancy in DebtMarket._transferDebt(address,uint256,address) (contracts/protocol/DebtMarket.sol#403-450):
        External calls:
        - ILendPool(vars.lendPoolAddress).updateReserveState(loanData.reserveAsset) (contracts/protocol/DebtMarket.sol#413)
        - ILendPool(vars.lendPoolAddress).updateReserveInterestRates(loanData.reserveAsset) (contracts/protocol/DebtMarket.sol#414)
        - IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.borrowAmount,reserveData.variableBorrowIndex) (contra
        - IDebtToken(reserveData.debtTokenAddress).mint(vars.buyer,vars.buyer,vars.borrowAmount,reserveData.variableBorrowIndex) (co
        - ILendPoolLoan(vars.lendPoolLoanAddress).reMintUNFT(loanData.nftAsset,loanData.nftTokenId,loanData.borrower,vars.buyer) (co
        State variables written after the call(s):
        - _deleteDebtOfferListing(nftAsset,tokenId) (contracts/protocol/DebtMarket.sol#449)
                - _totalDebtsByCollection[nftAsset] -= 1 (contracts/protocol/DebtMarket.sol#400)
        - _deleteDebtOfferListing(nftAsset,tokenId) (contracts/protocol/DebtMarket.sol#449)
```

```
                - _userTotalDebtByCollection[selldebt.debtor][nftAsset] -= 1 (contracts/protocol/DebtMarket.sol#399)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

ReserveLogic.getNormalizedIncome(DataTypes.ReserveData) (contracts/libraries/logic/ReserveLogic.sol#53-67) uses timestamp for compar
        Dangerous comparisons:
        - timestamp == uint40(block.timestamp) (contracts/libraries/logic/ReserveLogic.sol#57)
ReserveLogic.getNormalizedDebt(DataTypes.ReserveData) (contracts/libraries/logic/ReserveLogic.sol#76-90) uses timestamp for comparis
        Dangerous comparisons:
        - timestamp == uint40(block.timestamp) (contracts/libraries/logic/ReserveLogic.sol#80)
MathUtils.calculateCompoundedInterest(uint256,uint40,uint256) (contracts/libraries/math/MathUtils.sol#39-64) uses timestamp for compa
        Dangerous comparisons:
        - exp == 0 (contracts/libraries/math/MathUtils.sol#47)
        - exp > 2 (contracts/libraries/math/MathUtils.sol#53)
DebtMarket.createDebtListing(address,uint256,uint256,address,uint256,uint256) (contracts/protocol/DebtMarket.sol#142-188) uses times
        Dangerous comparisons:
        - require(bool,string)(auctionEndTimestamp >= block.timestamp,Errors.DM_AUCTION_ALREADY_ENDED) (contracts/protocol/DebtMarke
DebtMarket.bid(address,uint256,uint256,address) (contracts/protocol/DebtMarket.sol#277-333) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= marketListing.auctionEndTimestamp,Errors.DM_AUCTION_ALREADY_ENDED) (contracts/prot
DebtMarket.claim(address,uint256,address) (contracts/protocol/DebtMarket.sol#338-363) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > marketListing.auctionEndTimestamp,Errors.DM_AUCTION_NOT_ALREADY_ENDED) (contracts/p
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable._revert(bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#206-218) use
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#211-214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['0.8.4', '^0.8.0', '^0.8.1', '^0.8.2']
        - 0.8.4 (contracts/interfaces/IDebtMarket.sol#2)
        - 0.8.4 (contracts/interfaces/IDebtToken.sol#2)
        - 0.8.4 (contracts/interfaces/IIncentivesController.sol#2)
        - 0.8.4 (contracts/interfaces/IInterestRate.sol#2)
        - 0.8.4 (contracts/interfaces/ILendPool.sol#2)
        - 0.8.4 (contracts/interfaces/ILendPoolAddressesProvider.sol#2)
        - 0.8.4 (contracts/interfaces/ILendPoolLoan.sol#2)
        - 0.8.4 (contracts/interfaces/ILockeyManager.sol#2)
        - 0.8.4 (contracts/interfaces/INFTOracleGetter.sol#2)
        - 0.8.4 (contracts/interfaces/IReserveOracleGetter.sol#2)
        - 0.8.4 (contracts/interfaces/IScaledBalanceToken.sol#2)
        - 0.8.4 (contracts/interfaces/IUNFT.sol#2)
        - 0.8.4 (contracts/interfaces/IUToken.sol#2)
        - 0.8.4 (contracts/libraries/configuration/NftConfiguration.sol#2)
        - 0.8.4 (contracts/libraries/configuration/ReserveConfiguration.sol#2)
        - 0.8.4 (contracts/libraries/helpers/Errors.sol#2)
        - 0.8.4 (contracts/libraries/logic/GenericLogic.sol#2)
        - 0.8.4 (contracts/libraries/logic/ReserveLogic.sol#2)
        - 0.8.4 (contracts/libraries/math/MathUtils.sol#2)
        - 0.8.4 (contracts/libraries/math/PercentageMath.sol#2)
        - 0.8.4 (contracts/libraries/math/WadRayMath.sol#2)
        - 0.8.4 (contracts/libraries/types/DataTypes.sol#2)
        - 0.8.4 (contracts/protocol/DebtMarket.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721EnumerableUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721MetadataUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

AUTOMATED TESTING

```
AddressUpgradeable.functionCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#85-87)
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgrad
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#1
AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeabl
AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65) is
AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is never us
ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and
CountersUpgradeable.decrement(CountersUpgradeable.Counter) (node_modules/@openzeppelin/contracts-upgradeable/utils/CountersUpgradeabl
CountersUpgradeable.reset(CountersUpgradeable.Counter) (node_modules/@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.so
GenericLogic.calculateAvailableBorrows(uint256,uint256,uint256) (contracts/libraries/logic/GenericLogic.sol#239-252) is never used an
GenericLogic.calculateHealthFactorFromBalances(uint256,uint256,uint256) (contracts/libraries/logic/GenericLogic.sol#220-228) is never
GenericLogic.calculateLoanBidFine(address,DataTypes.ReserveData,address,DataTypes.NftConfigurationMap,DataTypes.LoanData,address,addre
GenericLogic.calculateLoanData(address,DataTypes.ReserveData,address,uint256,DataTypes.NftConfigurationMap,address,uint256,address,ad
GenericLogic.calculateLoanLiquidatePrice(uint256,address,DataTypes.ReserveData,address,uint256,DataTypes.NftConfigurationMap,address,
GenericLogic.calculateNftCollateralData(address,DataTypes.ReserveData,address,uint256,address,address) (contracts/libraries/logic/Gen
GenericLogic.calculateNftDebtData(address,DataTypes.ReserveData,address,uint256,address) (contracts/libraries/logic/GenericLogic.sol#
GenericLogic.calculateOptimalMaxRedeemValue(uint256,uint256) (contracts/libraries/logic/GenericLogic.sol#209-211) is never used and sh
GenericLogic.calculateOptimalMinRedeemValue(uint256,address,uint256,address,uint256,uint256) (contracts/libraries/logic/GenericLogic.s
Initializable._disableInitializers() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#144-150) is never
Initializable._getInitializedVersion() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#155-157) is ne
Initializable._isInitializing() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#162-164) is never use
MathUtils.calculateCompoundedInterest(uint256,uint40) (contracts/libraries/math/MathUtils.sol#71-73) is never used and should be remov
MathUtils.calculateCompoundedInterest(uint256,uint40,uint256) (contracts/libraries/math/MathUtils.sol#39-64) is never used and should
MathUtils.calculateLinearInterest(uint256,uint40) (contracts/libraries/math/MathUtils.sol#19-24) is never used and should be removed
NftConfiguration.getActive(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#121-123) is never us
NftConfiguration.getAuctionDuration(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#179-181) is
NftConfiguration.getAuctionParams(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#305-316) is n
NftConfiguration.getAuctionParamsMemory(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#338-347
NftConfiguration.getCollateralParams(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#288-298)
NftConfiguration.getCollateralParamsMemory(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#323-
NftConfiguration.getConfigTimestamp(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#259-261) is
NftConfiguration.getConfigTimestampMemory(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#363-3
NftConfiguration.getFlags(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#268-272) is never us
NftConfiguration.getFlagsMemory(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#279-281) is ne
NftConfiguration.getFrozen(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#139-141) is never u
NftConfiguration.getLiquidationBonus(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#103-105)
NftConfiguration.getLiquidationThreshold(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#83-85)
NftConfiguration.getLtv(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#63-65) is never used a
NftConfiguration.getMinBidFine(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#239-241) is nev
NftConfiguration.getMinBidFineMemory(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#354-356)
NftConfiguration.getRedeemDuration(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#159-161) is
NftConfiguration.getRedeemFine(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#199-201) is nev
NftConfiguration.getRedeemThreshold(DataTypes.NftConfigurationMap) (contracts/libraries/configuration/NftConfiguration.sol#219-221) i
NftConfiguration.setActive(DataTypes.NftConfigurationMap,bool) (contracts/libraries/configuration/NftConfiguration.sol#112-114) is ne
NftConfiguration.setAuctionDuration(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.sol#16
NftConfiguration.setConfigTimestamp(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.sol#24
NftConfiguration.setFrozen(DataTypes.NftConfigurationMap,bool) (contracts/libraries/configuration/NftConfiguration.sol#130-132) is ne
NftConfiguration.setLiquidationBonus(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.sol#9
NftConfiguration.setLiquidationThreshold(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.s
NftConfiguration.setLtv(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.sol#52-56) is neve
NftConfiguration.setMinBidFine(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.sol#228-232
NftConfiguration.setRedeemDuration(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.sol#148
NftConfiguration.setRedeemFine(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.sol#188-192
NftConfiguration.setRedeemThreshold(DataTypes.NftConfigurationMap,uint256) (contracts/libraries/configuration/NftConfiguration.sol#208
PercentageMath.percentDiv(uint256,uint256) (contracts/libraries/math/PercentageMath.sol#44-51) is never used and should be removed
ReserveConfiguration.getActive(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#133-135)
ReserveConfiguration.getBorrowingEnabled(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.s
ReserveConfiguration.getDecimals(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#115-1
ReserveConfiguration.getFlags(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#218-236)
ReserveConfiguration.getFlagsMemory(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#29
ReserveConfiguration.getFrozen(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#151-153)
ReserveConfiguration.getLiquidationBonus(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.s
ReserveConfiguration.getLiquidationThreshold(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfigurati
ReserveConfiguration.getLtv(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#55-57) is n
ReserveConfiguration.getParams(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#243-263)
ReserveConfiguration.getParamsMemory(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#2
ReserveConfiguration.getReserveFactor(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfiguration.sol#
ReserveConfiguration.getStableRateBorrowingEnabled(DataTypes.ReserveConfigurationMap) (contracts/libraries/configuration/ReserveConfig
ReserveConfiguration.setActive(DataTypes.ReserveConfigurationMap,bool) (contracts/libraries/configuration/ReserveConfiguration.sol#12
```

```
ReserveConfiguration.setBorrowingEnabled(DataTypes.ReserveConfigurationMap,bool) (contracts/libraries/configuration/ReserveConfigurat
ReserveConfiguration.setDecimals(DataTypes.ReserveConfigurationMap,uint256) (contracts/libraries/configuration/ReserveConfiguration.s
ReserveConfiguration.setFrozen(DataTypes.ReserveConfigurationMap,bool) (contracts/libraries/configuration/ReserveConfiguration.sol#14
ReserveConfiguration.setLiquidationBonus(DataTypes.ReserveConfigurationMap,uint256) (contracts/libraries/configuration/ReserveConfigu
ReserveConfiguration.setLiquidationThreshold(DataTypes.ReserveConfigurationMap,uint256) (contracts/libraries/configuration/ReserveCon
ReserveConfiguration.setLtv(DataTypes.ReserveConfigurationMap,uint256) (contracts/libraries/configuration/ReserveConfiguration.sol#44
ReserveConfiguration.setReserveFactor(DataTypes.ReserveConfigurationMap,uint256) (contracts/libraries/configuration/ReserveConfigurat
ReserveConfiguration.setStableRateBorrowingEnabled(DataTypes.ReserveConfigurationMap,bool) (contracts/libraries/configuration/Reserve
ReserveLogic._mintToTreasury(DataTypes.ReserveData,uint256,uint256,uint256,uint256,uint40) (contracts/libraries/logic/ReserveLogic.sc
ReserveLogic._updateIndexes(DataTypes.ReserveData,uint256,uint256,uint256,uint40) (contracts/libraries/logic/ReserveLogic.sol#272-308
ReserveLogic.cumulateToLiquidityIndex(DataTypes.ReserveData,uint256,uint256) (contracts/libraries/logic/ReserveLogic.sol#126-139) is
ReserveLogic.getNormalizedDebt(DataTypes.ReserveData) (contracts/libraries/logic/ReserveLogic.sol#76-90) is never used and should be
ReserveLogic.getNormalizedIncome(DataTypes.ReserveData) (contracts/libraries/logic/ReserveLogic.sol#53-67) is never used and should b
ReserveLogic.updateInterestRates(DataTypes.ReserveData,address,address,uint256,uint256) (contracts/libraries/logic/ReserveLogic.sol#1
ReserveLogic.updateState(DataTypes.ReserveData) (contracts/libraries/logic/ReserveLogic.sol#96-118) is never used and should be remov
SafeERC20Upgradeable.safeApprove(IERC20Upgradeable,address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/uti
SafeERC20Upgradeable.safeDecreaseAllowance(IERC20Upgradeable,address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/toker
SafeERC20Upgradeable.safeIncreaseAllowance(IERC20Upgradeable,address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/toker
SafeERC20Upgradeable.safePermit(IERC20PermitUpgradeable,address,address,uint256,uint256,uint8,bytes32,bytes32) (node_modules/@openzep
WadRayMath.halfRay() (contracts/libraries/math/WadRayMath.sol#39-41) is never used and should be removed
WadRayMath.halfWad() (contracts/libraries/math/WadRayMath.sol#46-48) is never used and should be removed
WadRayMath.rayDiv(uint256,uint256) (contracts/libraries/math/WadRayMath.sol#103-110) is never used and should be removed
WadRayMath.rayMul(uint256,uint256) (contracts/libraries/math/WadRayMath.sol#87-95) is never used and should be removed
WadRayMath.rayToWad(uint256) (contracts/libraries/math/WadRayMath.sol#117-123) is never used and should be removed
WadRayMath.wad() (contracts/libraries/math/WadRayMath.sol#32-34) is never used and should be removed
WadRayMath.wadDiv(uint256,uint256) (contracts/libraries/math/WadRayMath.sol#72-79) is never used and should be removed
WadRayMath.wadMul(uint256,uint256) (contracts/libraries/math/WadRayMath.sol#56-64) is never used and should be removed
WadRayMath.wadToRay(uint256) (contracts/libraries/math/WadRayMath.sol#130-134) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4) allows
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#4) al
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#4) allows old versi
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol#4) allows old vers
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721EnumerableUpgradeable.sol#4) al
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721MetadataUpgradeable.sol#4) allo
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4) allows old versi
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version0.8.4 (contracts/interfaces/IDebtMarket.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/IDebtToken.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/IIncentivesController.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/IInterestRate.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/ILendPool.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/ILendPoolLoan.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/ILockeyManager.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/INFTOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/IReserveOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/IScaledBalanceToken.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/IUNFT.sol#2) allows old versions
Pragma version0.8.4 (contracts/interfaces/IUToken.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/configuration/NftConfiguration.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/configuration/ReserveConfiguration.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/helpers/Errors.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/logic/GenericLogic.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/logic/ReserveLogic.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/math/MathUtils.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/math/PercentageMath.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/math/WadRayMath.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/types/DataTypes.sol#2) allows old versions
Pragma version0.8.4 (contracts/protocol/DebtMarket.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Pragma version0.8.4 (contracts/libraries/math/WadRayMath.sol#2) allows old versions
Pragma version0.8.4 (contracts/libraries/types/DataTypes.sol#2) allows old versions
Pragma version0.8.4 (contracts/protocol/DebtMarket.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgrad
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgrad
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgra
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.s
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IE
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedC
Parameter ILendPool.liquidateSudoSwap(address,uint256,uint256,address,uint256).LSSVMPair (contracts/interfaces/ILendPool.sol#375) is
Function IUToken.UNDERLYING_ASSET_ADDRESS() (contracts/interfaces/IUToken.sol#149) is not in mixedCase
Function IUToken.RESERVE_TREASURY_ADDRESS() (contracts/interfaces/IUToken.sol#154) is not in mixedCase
Variable DebtMarket._addressesProvider (contracts/protocol/DebtMarket.sol#38) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "reserveData (contracts/libraries/logic/GenericLogic.sol#158)" inGenericLogic (contracts/libraries/logic/Generic
Redundant expression "nftAsset (contracts/libraries/logic/GenericLogic.sol#340)" inGenericLogic (contracts/libraries/logic/GenericLog
Redundant expression "timestamp (contracts/libraries/logic/ReserveLogic.sol#240)" inReserveLogic (contracts/libraries/logic/ReserveLo
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Variable Errors.LPL_BID_AUCTION_DURATION_HAS_END (contracts/libraries/helpers/Errors.sol#98) is too similar to Errors.LPL_BID_AUCTION
Variable Errors.RC_INVALID_MAX_BID_FINE (contracts/libraries/helpers/Errors.sol#146) is too similar to Errors.RC_INVALID_MIN_BID_FINE
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

NftConfiguration.slitherConstructorConstantVariables() (contracts/libraries/configuration/NftConfiguration.sol#12-366) uses literals
        - CONFIG_TIMESTAMP_MASK = 0xFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF (contracts/libraries/configurati
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

PercentageMath.ONE_PERCENT (contracts/libraries/math/PercentageMath.sol#17) is never used in PercentageMath (contracts/libraries/math
PercentageMath.TEN_PERCENT (contracts/libraries/math/PercentageMath.sol#18) is never used in PercentageMath (contracts/libraries/math
PercentageMath.ONE_THOUSANDTH_PERCENT (contracts/libraries/math/PercentageMath.sol#19) is never used in PercentageMath (contracts/lib
PercentageMath.ONE_TEN_THOUSANDTH_PERCENT (contracts/libraries/math/PercentageMath.sol#20) is never used in PercentageMath (contracts
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

- As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives.

6.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX Results:

- No issues found by Mythx.

THANK YOU FOR CHOOSING

// HALBORN