HALBORN

Unlockd Finance - Protocol v1 - Updated

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 10th, 2022 - November 30th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/22/2022	István Böhm
0.2	Document Updates	12/01/2022	István Böhm
0.3	Final Draft	12/01/2022	István Böhm
0.4	Draft Review	12/01/2022	Roberto Reigada
0.5	Draft Review	12/01/2022	Piotr Cielas
0.6	Draft Review	12/01/2022	Gabi Urrutia
1.0	Remediation Plan	12/20/2022	István Böhm
1.1	Remediation Plan Review	12/21/2022	Roberto Reigada
1.2	Remediation Plan Review	12/21/2022	Piotr Cielas
1.3	Remediation Plan Review	12/21/2022	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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com
István Böhm	Halborn	Istvan.Bohm@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Unlockd is a decentralized noncustodial NFT lending protocol where users can participate as depositors or borrowers.

Unlockd Finance engaged Halborn to conduct a security audit on their smart contracts beginning on November 10th, 2022 and ending on November 30th, 2022. The security assessment was scoped to specific smart contracts provided in the UnlockdFinance/unlockd-protocol-v1 GitHub repository. The commit hash and further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

The team at Halborn was provided three 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:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the Unlockd Finance team. The main ones were the following:

- Fix the price calculation formula in the `getNFTPriceNFTX` function of the `NFTOracle` contract.
- Include constructors and automatically mark the upgradeable contracts as initialized.
- Revert the `triggerUserCollateral` function of the `LendPool` contract if its `nftAsset` parameter is invalid.

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 code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

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

Code repositories:

1. Unlockd Protocol Smart Contracts:

- Repository: UnlockdFinance/unlockd-protocol-v1
- Commit ID: c0fe7285470728c4971e054519ce3e6fc542cdd5
- Smart contracts in scope:
  - contracts/libraries/logic/LiquidateLogic.sol
  - contracts/libraries/logic/LiquidateMarketsLogic.sol
  - contracts/libraries/nftx/NFTXHelper.sol
  - contracts/misc/UnlockdProtocolDataProvider.sol
  - contracts/protocol/LendPool.sol
  - contracts/protocol/LendPoolAddressesProvider.sol
  - contracts/protocol/LendPoolConfigurator.sol
  - contracts/protocol/LendPoolLoan.sol
  - contracts/protocol/NFTOracle.sol
  - contracts/protocol/PunkGateway.sol
  - contracts/protocol/WETHGateway.sol

Out-of-scope:

- Unlockd protocol contracts outside the scope
- Third-party libraries and dependencies
- Economic attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	6

LIKELIHOOD

IMPACT

		(HAL-01)		
(HAL-04) (HAL-05)	(HAL-02) (HAL-03)			
(HAL-06) (HAL-07) (HAL-08) (HAL-09)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - IMPROPER NFTORACLE NFTX PRICE CALCULATION	Medium	SOLVED - 11/30/2022
HAL-02 - UNINITIALIZED IMPLEMENTATION CONTRACTS	Low	SOLVED - 12/05/2022
HAL-03 - LACK OF ASSET VALIDATION IN TRIGGERUSERCOLLATERAL	Low	SOLVED - 12/12/2022
HAL-04 - INCOMPLETE NFTXHELPER NFTX PRICE CALCULATION	Informational	ACKNOWLEDGED
HAL-05 - NFTS SENT BY MISTAKE CAN NOT BE RETRIEVED	Informational	SOLVED - 12/05/2022
HAL-06 - REDUNDANT EXPRESSIONS	Informational	SOLVED - 12/05/2022
HAL-07 - UNUSED LIBRARIES	Informational	SOLVED - 12/05/2022
HAL-08 - MISSING/INCOMPLETE NATSPEC COMMENTS	Informational	ACKNOWLEDGED
HAL-09 - MISLEADING CODE COMMENTS	Informational	SOLVED - 12/05/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) IMPROPER NFTORACLE NFTX PRICE CALCULATION - MEDIUM

Description:

The getNFTPriceNFTX function in the NFTOracle contract does not calculate the price correctly. The amountIn value is calculated with an invalid formula, and the minting price of the vault is not subtracted from it.

In the current version of the NFTXVaultUpgradeable contract, the decimals of the vault tokens are constant and equal to 18.

Code Location:

Listing 1: NFTOracle.sol (Line 293)

```
287 // Swap path is NFTX Vault -> WETH
288 address[] memory swapPath = new address[](2);
289 swapPath[0] = address(nftxVault);
290 swapPath[1] = IUniswapV2Router02(sushiswapRouter).WETH();
291
292 // Get the price from sushiswap
293 uint256 amountIn = 1**IERC20MetadataUpgradeable(address(nftxVault)
 ↳ ).decimals();
294 uint256[] memory amounts = IUniswapV2Router02(sushiswapRouter).
 ↳ getAmountsOut(amountIn, swapPath);
295 return amounts[1];
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

The price calculation formula should be fixed and the minting fee should be subtracted from the value as given in the following snippet:

Listing 2: NFTOracle.sol

```
// Calculate the minting fee
(uint256 mintFee, , , , ) = INFTXVaultFactoryV2(nftxVaultFactory).
↳ vaultFees(nftxVault.vaultId());
// Get the price from sushiswap
uint256 amountIn = 1 ether - mintFee;
uint256[] memory amounts = IUniswapV2Router02(sushiswapRouter).
↳ getAmountsOut(amountIn, swapPath);
```

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 936c56f by removing the getNFTPriceNFTX function from the NFTOracle contract.

3.2 (HAL-02) UNINITIALIZED IMPLEMENTATION CONTRACTS - LOW

Description:

Multiple contracts are using the Initializable module from OpenZeppelin, and the implementations of these contracts are not initialized by the protocol. In the proxy pattern, an uninitialized implementation contract can be initialized by someone else to take over the contract. Even if it does not affect the proxy contracts directly, it is a good practice to initialize them to prevent any mishap against unseen vulnerabilities.

In the latest version (4.8.0), this is done by calling the _disableInitializers function in the constructor. However, in the currently used version (4.4.1), this is done by adding an empty constructor with initializer modifier to the upgradable contracts.

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider including a constructor to automatically mark the upgradeable contracts as initialized when they are deployed:

Listing 3: Initialization Example

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() initializer {}
```

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit d7f763a by marking upgradeable contracts as initialized when deployed.

3.3 (HAL-03) LACK OF ASSET VALIDATION IN TRIGGERUSERCOLLATERAL - LOW

Description:

In the LendPool contract, users can trigger the valuation and configuration of their NFTs by calling the triggerUserCollateral function.

It was identified that this function does not validate whether the asset is configured with the protocol or not. Users who call this function with an NFT asset that is not configured with the protocol would lose their configuration fee, even though their token would not be configured and evaluated as expected.

Code Location:

Listing 4: Initialization Example

```
  function triggerUserCollateral(address nftAsset, uint256
↳ nftTokenId)
    external
    payable
    override
    onlyHolder(nftAsset, nftTokenId)
    whenNotPaused
  {
    require(_configFee == msg.value);
    emit UserCollateralTriggered(_msgSender(), nftAsset,
↳ nftTokenId);
  }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider validating the nftAsset parameter and revert the function if the asset is not configured in the protocol.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 203aa1b and 062314b by adding the onlyCollection modifier and validating the nftAsset parameter.

3.4 (HAL-04) INCOMPLETE NFTXHELPER NFTX PRICE CALCULATION - INFORMATIONAL

Description:

It was identified that the getNFTXPrice function in the NFTXHelper contract reverts if there are multiple NTFX vaults associated with the queried asset. In those unlikely cases, the function would not be able to calculate token prices for the affected assets.

Code Location:

Listing 5: NFTXHelper.sol (Lines 104,110)

```
102     address[] memory vaultAddresses = INFTXVaultFactoryV2(
 ↳ vaultFactoryAddress).vaultsForAsset(nftAsset);
103
104     require(vaultAddresses.length > 0, Errors.
 ↳ NFTX_INVALID_VAULTS_LENGTH);
105
106     uint256[] memory tokenIds = new uint256[](1);
107     tokenIds[0] = nftTokenId;
108
109     // Always get the first vault address
110     address vaultAddress = vaultAddresses[0];
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to make the price calculation more flexible by checking the eligibility in multiple vaults.

Remediation Plan:

ACKNOWLEDGED: The Unlockd Finance team acknowledged this finding. Based on their research, only one vault is created per collection, and it is still possible to upgrade the contracts in case of possible future changes.

3.5 (HAL-05) NFTS SENT BY MISTAKE CAN NOT BE RETRIEVED - INFORMATIONAL

Description:

It was identified that the LendPool contract is missing the function to sweep/retrieve the NFTs send to the contract mistakenly. Such NFTs are locked in the contract indefinitely. Currently, it is only possible to sweep accidentally transferred ERC-20 tokens.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Consider also adding a function to sweep NFTs send to the LendPool contract mistakenly.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 49947b2 by adding the rescueNFT function to sweep NFTs sent to LendPool contract mistakenly.

3.6 (HAL-06) REDUNDANT EXPRESSIONS - INFORMATIONAL

Description:

It was identified that the contracts have multiple lines of unused or redundant code:

Code Location:

Listing 6: NFTOracle.sol (Lines 34-38)

```
34  /**
35   * @dev Emitted when the admin has been updated
36   * @param admin The new admin
37   **/
38  event FeedAdminUpdated(address indexed admin);
```

Listing 7: NFTOracle.sol (Lines 93,103-106)

```
92  function initialize(
93    address _admin,
94    address _nftxVaultFactory,
95    address _sushiswapRouter,
96    address _lendPoolConfigurator
97  ) public initializer {
98    require(
99      _admin != address(0) && _nftxVaultFactory != address(0) &&
 ↳ _sushiswapRouter != address(0),
100     Errors.INVALID_ZERO_ADDRESS
101   );
102   __Ownable_init();
103   require(
104     _admin != address(0) && _nftxVaultFactory != address(0) &&
 ↳ _sushiswapRouter != address(0),
105     Errors.INVALID_ZERO_ADDRESS
106   );
```

Listing 8: LendPoolAddressesProvider.sol (Line 30)

```
30   bytes32 private constant UNLOCKD_ORACLE = "UNLOCKD_ORACLE";
```

Listing 9: LendPool.sol (Lines 420-423)

```
414  function onERC721Received(
415    address operator,
416    address from,
417    uint256 tokenId,
418    bytes calldata data
419  ) external pure override returns (bytes4) {
420    operator;
421    from;
422    tokenId;
423    data;
```

Listing 10: LendPool.sol (Lines 737-742)

```
729  function finalizeTransfer(
730    address asset,
731    address from,
732    address to,
733    uint256 amount,
734    uint256 balanceFromBefore,
735    uint256 balanceToBefore
736  ) external view override whenNotPaused {
737    asset;
738    from;
739    to;
740    amount;
741    balanceFromBefore;
742    balanceToBefore;
```

Listing 11: LendPoolLoan.sol (Lines 367-370)

```
361   function onERC721Received(
362     address operator,
363     address from,
364     uint256 tokenId,
365     bytes calldata data
366   ) external pure override returns (bytes4) {
367     operator;
368     from;
369     tokenId;
370     data;
371     return IERC721ReceiverUpgradeable.onERC721Received.selector;
372   }
```

Listing 12: UnlockdProtocolDataProvider.sol (Line 19)

```
19   address constant ETH = 0
↳ xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to review the contracts and remove any unnecessary code.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 49947b2 by removing unnecessary code.

3.7 (HAL-07) UNUSED LIBRARIES - INFORMATIONAL

Description:

Multiple unused library imports were identified in the contracts:

contracts/protocol/NFTOracle.sol:
- BlockContext

contracts/protocol/LendPool.sol:
- IERC721Upgradeable (included twice)
- WadRayMath
- PercentageMath

contracts/protocol/LendPoolConfigurator.sol:
- IUNFT

contracts/libraries/logic/LiquidateLogic.sol:
- SushiSwapHelper
- WadRayMath
- MathUtils

contracts/libraries/logic/LiquidateMarketsLogic.sol:
- MathUtils
- WadRayMath
- SushiSwapHelper
- IERC721Upgradeable
- IERC721MetadataUpgradeable

Unused imports decrease the readability of the contracts.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to review the contracts and remove any unnecessary imports from them.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 4f3d541 by removing unnecessary imports.

3.8 (HAL-08) MISSING/INCOMPLETE NATSPEC COMMENTS - INFORMATIONAL

Description:

It was identified that some functions have missing or incomplete code documentation, which affects the understandability, auditability, and usability of the code.

Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables, and more. This special form is named the Ethereum Natural Language Specification Format (*NatSpec**).

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding full **NatSpec** comments so that all functions have full code documentation for future use.

Remediation Plan:

ACKNOWLEDGED: The Unlockd Finance team acknowledged this finding, but will add additional documentation to the code in future releases.

3.9 (HAL-09) MISLEADING CODE COMMENTS - INFORMATIONAL

Description:

It was identified that multiple functions in the contracts feature misleading comments. While comments are useful for understanding the true purpose and functionality of the code, misleading comments have been detected that do not match the actual implementation in the code. The following section contains two examples to demonstrate the issue.

Code Location:

The _depositPunk function has an inappropriate description accidentally copied from the _repay function:

Listing 13: PunkGateway.sol (Line 151)

```
149   /**
150    * @notice Deposits a punk given its index
151    * - E.g. User repays 100 USDC, burning loan and receives
 ↳ collateral asset
152    * @param punkIndex The index of the CryptoPunk to deposit
153    **/
154   function _depositPunk(uint256 punkIndex) internal {
```

The getNFTXPrice function returns the price in the currency of the associated reserve, which is not necessarily ETH:

Listing 14: NFTXHelper.sol (Line 87)

```
86   /**
87    * @dev Get the NFTX price in ETH
88    * @param addressesProvider The addresses provider
89    * @param nftAsset The underlying NFT address
90    * @param nftTokenId The underlying NFT token Id
91    */
92   function getNFTXPrice(
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to review the contracts and correct any comments that do not accurately describe the actual implementation.

Note that this finding only listed two examples. However, the comments in all the contracts should be reviewed and fixed if necessary.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 19ffa6f by correcting the comments above.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

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:

contracts/libraries/logic/LiquidateLogic.sol

```
LiquidateLogic.executeRedeem(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteRedeemParams).vars (contracts/libraries/logic/LiquidateLogic.sol#266) is a local variable never initialized
LiquidateLogic.executeLiquidate(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteLiquidateParams).vars (contracts/libraries/logic/LiquidateLogic.sol#406) is a local variable never initialized
LiquidateLogic.executeAuction(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteAuctionParams).vars (contracts/libraries/logic/LiquidateLogic.sol#135) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Reentrancy in LiquidateLogic.executeAuction(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteAuctionParams) (contracts/libraries/logic/LiquidateLogic.sol#125-231):
        External calls:
        - reserveData.updateState() (contracts/libraries/logic/LiquidateLogic.sol#159)
        - ILendPoolLoan(vars.loanAddress).auctionLoan(vars.initiator,vars.loanId,params.onBehalfOf,params.bidPrice,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#201-208)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,address(this),params.bidPrice) (contracts/libraries/logic/LiquidateLogic.sol#211)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.bidderAddress,loanData.bidPrice) (contracts/libraries/logic/LiquidateLogic.sol#215)
        Event emitted after the call(s):
        - Auction(vars.initiator,loanData.reserveAsset,params.bidPrice,params.nftAsset,params.nftTokenId,params.onBehalfOf,loanData.borrower,vars.loanId) (contracts/libraries/logic/LiquidateLogic.sol#221-230)
Reentrancy in LiquidateLogic.executeLiquidate(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteLiquidateParams) (contracts/libraries/logic/LiquidateLogic.sol#398-518):
        External calls:
        - reserveData.updateState() (contracts/libraries/logic/LiquidateLogic.sol#434)
        - ILendPoolLoan(vars.poolLoan).liquidateLoan(loanData.bidderAddress,vars.loanId,nftData.uNftAddress,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#458-464)
        - IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#466-470)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,address(this),vars.extraDebtAmount) (contracts/libraries/logic/LiquidateLogic.sol#477)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransfer(reserveData.uTokenAddress,vars.borrowAmount) (contracts/libraries/logic/LiquidateLogic.sol#481)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.borrower,vars.remainAmount) (contracts/libraries/logic/LiquidateLogic.sol#485)
        - (success) = address(loanData.nftAsset).call(abi.encodeWithSignature(safeTransferFrom(address,address,uint256),address(this),loanData.bidderAddress,params.nftTokenId)) (contracts/libraries/logic/LiquidateLogic.sol#490-497)
        - IERC721Upgradeable(loanData.nftAsset).safeTransferFrom(address(this),IUToken(reserveData.uTokenAddress).RESERVE_TREASURY_ADDRESS(),params.nftTokenId) (contracts/libraries/logic/LiquidateLogic.sol#500-504)
        Event emitted after the call(s):
        - Liquidate(vars.initiator,loanData.reserveAsset,vars.borrowAmount,vars.remainAmount,loanData.nftAsset,loanData.nftTokenId,loanData.borrower,vars.loanId) (contracts/libraries/logic/LiquidateLogic.sol#506-515)
Reentrancy in LiquidateLogic.executeRedeem(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteRedeemParams) (contracts/libraries/logic/LiquidateLogic.sol#258-374):
        External calls:
        - reserveData.updateState() (contracts/libraries/logic/LiquidateLogic.sol#294)
        - ILendPoolLoan(vars.poolLoan).redeemLoan(vars.initiator,vars.loanId,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#331-336)
        - IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#338)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,reserveData.uTokenAddress,vars.repayAmount) (contracts/libraries/logic/LiquidateLogic.sol#344-348)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.bidderAddress,loanData.bidPrice) (contracts/libraries/logic/LiquidateLogic.sol#352)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,loanData.firstBidderAddress,vars.bidFine) (contracts/libraries/logic/LiquidateLogic.sol#355-359)
        Event emitted after the call(s):
        - Redeem(vars.initiator,loanData.reserveAsset,vars.repayAmount,vars.bidFine,loanData.nftAsset,loanData.nftTokenId,loanData.borrower,vars.loanId) (contracts/libraries/logic/LiquidateLogic.sol#362-371)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

LiquidateLogic.executeAuction(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteAuctionParams) (contracts/libraries/logic/LiquidateLogic.sol#125-231) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(block.timestamp <= vars.auctionEndTimestamp,Errors.LPL_BID_AUCTION_DURATION_HAS_END) (contracts/libraries/logic/LiquidateLogic.sol#194)
LiquidateLogic.executeRedeem(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteRedeemParams) (contracts/libraries/logic/LiquidateLogic.sol#258-374) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(block.timestamp <= vars.redeemEndTimestamp,Errors.LPL_BID_REDEEM_DURATION_HAS_END) (contracts/libraries/logic/LiquidateLogic.sol#291)
LiquidateLogic.executeLiquidate(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteLiquidateParams) (contracts/libraries/logic/LiquidateLogic.sol#398-518) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(block.timestamp > vars.auctionEndTimestamp,Errors.LPL_BID_AUCTION_DURATION_NOT_END) (contracts/libraries/logic/LiquidateLogic.sol#431)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Low level call in LiquidateLogic.executeLiquidate(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLendPoolStates,DataTypes.ExecuteLiquidateParams) (contracts/libraries/logic/LiquidateLogic.sol#398-518):
- (success) = address(loanData.nftAsset).call(abi.encodeWithSignature(safeTransferFrom(address,address,uint256),address(this),loanData.bidderAddress,params.nftTokenId)) (contracts/libraries/logic/LiquidateLogic.sol#490-497)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

contracts/libraries/logic/LiquidateMarketsLogic.sol

LiquidateMarketsLogic.executeLiquidateNFTX(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),mapping(address => bool),DataTypes.ExecuteLiquidateNFTXParams).vars (contracts/libraries/logic/LiquidateMarketsLogic.sol#88) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Reentrancy in LiquidateMarketsLogic.executeLiquidateNFTX(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),mapping(address => bool),DataTypes.ExecuteLiquidateNFTXParams) (contracts/libraries/logic/LiquidateMarketsLogic.sol#80-203):
External calls:
- reserveData.updateState() (contracts/libraries/logic/LiquidateMarketsLogic.sol#127)
- priceNFTX = ILendPoolLoan(vars.poolLoan).liquidateLoanNFTX(vars.loanId,nftData.uNftAddress,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateMarketsLogic.sol#141-146)
- IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateMarketsLogic.sol#162-166)
- IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(treasury,address(this),vars.extraDebtAmount) (contracts/libraries/logic/LiquidateMarketsLogic.sol#178)
- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(reserveData.uTokenAddress,vars.borrowAmount) (contracts/libraries/logic/LiquidateMarketsLogic.sol#182)
- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(vars.liquidator,vars.feeAmount) (contracts/libraries/logic/LiquidateMarketsLogic.sol#185)
- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.borrower,vars.remainAmount) (contracts/libraries/logic/LiquidateMarketsLogic.sol#189)
Event emitted after the call(s):
- LiquidateNFTX(loanData.reserveAsset,vars.borrowAmount,vars.remainAmount,loanData.nftAsset,loanData.nftTokenId,loanData.borrower,vars.loanId) (contracts/libraries/logic/LiquidateMarketsLogic.sol#192-200)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

contracts/libraries/nftx/NFTXHelper.sol

NFTXHelper.sellNFTX(ILendPoolAddressesProvider,address,uint256,address) (contracts/libraries/nftx/NFTXHelper.sol#31-84) ignores return value by nftxVault.mint(tokenIds,new uint256[](1)) (contracts/libraries/nftx/NFTXHelper.sol#56)
NFTXHelper.sellNFTX(ILendPoolAddressesProvider,address,uint256,address) (contracts/libraries/nftx/NFTXHelper.sol#31-84) ignores return value by IERC20Upgradeable(vaultAddress).approve(sushiSwapRouterAddress,depositAmount) (contracts/libraries/nftx/NFTXHelper.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

NFTXHelper.getNFTXPrice(ILendPoolAddressesProvider,address,uint256,address) (contracts/libraries/nftx/NFTXHelper.sol#92-136) is never used and should be removed
NFTXHelper.sellNFTX(ILendPoolAddressesProvider,address,uint256,address) (contracts/libraries/nftx/NFTXHelper.sol#31-84) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

contracts/misc/UnlockdProtocolDataProvider.sol

UnlockdProtocolDataProvider.getAllReservesTokenDatas() (contracts/misc/UnlockdProtocolDataProvider.sol#46-67) has external calls inside a loop: reserveData = pool.getReserveData(reserves[i]) (contracts/misc/UnlockdProtocolDataProvider.sol#52)
UnlockdProtocolDataProvider.getAllReservesTokenDatas() (contracts/misc/UnlockdProtocolDataProvider.sol#46-67) has external calls inside a loop: reservesTokens[i] = ReserveTokenData(IERC20Detailed(reserves[i]).symbol(),reserves[i],IERC20Detailed(reserveData.uTokenAddress).symbol(),reserveData.uTokenAddress,IERC20Detailed(reserveData.debtTokenAddress).symbol(),reserveData.debtTokenAddress) (contracts/misc/UnlockdProtocolDataProvider.sol#53-60)
UnlockdProtocolDataProvider.getAllNftsTokenDatas() (contracts/misc/UnlockdProtocolDataProvider.sol#90-109) has external calls inside a loop: nftData = pool.getNftData(nfts[i]) (contracts/misc/UnlockdProtocolDataProvider.sol#96)
UnlockdProtocolDataProvider.getAllNftsTokenDatas() (contracts/misc/UnlockdProtocolDataProvider.sol#90-109) has external calls inside a loop: nftTokens[i] = NftTokenData(IERC721Detailed(nfts[i]).symbol(),nfts[i],IERC721Detailed(nftData.uNftAddress).symbol(),nftData.uNftAddress) (contracts/misc/UnlockdProtocolDataProvider.sol#97-102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

UnlockdProtocolDataProvider.ETH (contracts/misc/UnlockdProtocolDataProvider.sol#19) is never used in UnlockdProtocolDataProvider (contracts/misc/UnlockdProtocolDataProvider.sol#15-335)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

contracts/protocol/LendPool.sol

```
LendPool.rescue(IERC20,address,uint256,bool) (contracts/protocol/LendPool.sol#1027-1039) sends eth to arbitrary user
        Dangerous calls:
        - (sent) = to.call{value: amount}() (contracts/protocol/LendPool.sol#1034)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

LendPool.getNftLiquidatePrice(address,uint256).vars (contracts/protocol/LendPool.sol#687) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

LendPool.rescue(IERC20,address,uint256,bool).to (contracts/protocol/LendPool.sol#1029) lacks a zero-check on :
                - (sent) = to.call{value: amount}() (contracts/protocol/LendPool.sol#1034)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Low level call in LendPool.rescue(IERC20,address,uint256,bool) (contracts/protocol/LendPool.sol#1027-1039):
        - (sent) = to.call{value: amount}() (contracts/protocol/LendPool.sol#1034)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Redundant expression "operator (contracts/protocol/LendPool.sol#420)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "from (contracts/protocol/LendPool.sol#421)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "tokenId (contracts/protocol/LendPool.sol#422)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "data (contracts/protocol/LendPool.sol#423)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "asset (contracts/protocol/LendPool.sol#737)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "from (contracts/protocol/LendPool.sol#738)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "to (contracts/protocol/LendPool.sol#739)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "amount (contracts/protocol/LendPool.sol#740)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "balanceFromBefore (contracts/protocol/LendPool.sol#741)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Redundant expression "balanceToBefore (contracts/protocol/LendPool.sol#742)" inLendPool (contracts/protocol/LendPool.sol#53-1092)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

contracts/protocol/LendPoolAddressesProvider.sol

```
Reentrancy in LendPoolAddressesProvider._updateImpl(bytes32,address) (contracts/protocol/LendPoolAddressesProvider.sol#403-420):
        External calls:
        - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#410)
        State variables written after the call(s):
        - _addresses[id] = address(proxy) (contracts/protocol/LendPoolAddressesProvider.sol#412)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

LendPoolAddressesProvider.setAddressAsProxy(bytes32,address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#70-82) ignores return value by Address.functionCa
ll(_addresses[id],encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#80)
LendPoolAddressesProvider.setLendPoolImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#118-126) ignores return value by Address.functionCall(_addr
esses[LEND_POOL],encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#124)
LendPoolAddressesProvider.setLendPoolConfiguratorImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#142-150) ignores return value by Address.functi
onCall(_addresses[LEND_POOL_CONFIGURATOR],encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#148)
LendPoolAddressesProvider.setLendPoolLoanImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#238-246) ignores return value by Address.functionCall(_
addresses[LEND_POOL_LOAN],encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#244)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Reentrancy in LendPoolAddressesProvider._updateImpl(bytes32,address) (contracts/protocol/LendPoolAddressesProvider.sol#403-420):
        External calls:
        - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#410)
        Event emitted after the call(s):
        - ProxyCreated(id,address(proxy)) (contracts/protocol/LendPoolAddressesProvider.sol#413)
Reentrancy in LendPoolAddressesProvider.setAddressAsProxy(bytes32,address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#70-82):
        External calls:
        - _updateImpl(id,implementationAddress) (contracts/protocol/LendPoolAddressesProvider.sol#76)
                - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#410)
                - proxy_scope_0.upgradeTo(newAddress) (contracts/protocol/LendPoolAddressesProvider.sol#418)
        Event emitted after the call(s):
        - AddressSet(id,implementationAddress,true,encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#77)

Reentrancy in LendPoolAddressesProvider.setLendPoolConfiguratorImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#142-150):
        External calls:
        - _updateImpl(LEND_POOL_CONFIGURATOR,configurator) (contracts/protocol/LendPoolAddressesProvider.sol#144)
                - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#410)
                - proxy_scope_0.upgradeTo(newAddress) (contracts/protocol/LendPoolAddressesProvider.sol#418)
        Event emitted after the call(s):
        - LendPoolConfiguratorUpdated(configurator,encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#145)
Reentrancy in LendPoolAddressesProvider.setLendPoolImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#118-126):
        External calls:
        - _updateImpl(LEND_POOL,pool) (contracts/protocol/LendPoolAddressesProvider.sol#120)
                - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#410)
                - proxy_scope_0.upgradeTo(newAddress) (contracts/protocol/LendPoolAddressesProvider.sol#418)
        Event emitted after the call(s):
        - LendPoolUpdated(pool,encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#121)
Reentrancy in LendPoolAddressesProvider.setLendPoolLoanImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#238-246):
        External calls:
        - _updateImpl(LEND_POOL_LOAN,loanAddress) (contracts/protocol/LendPoolAddressesProvider.sol#240)
                - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#410)
                - proxy_scope_0.upgradeTo(newAddress) (contracts/protocol/LendPoolAddressesProvider.sol#418)
        Event emitted after the call(s):
        - LendPoolLoanUpdated(loanAddress,encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#241)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

LendPoolAddressesProvider.UNLOCKD_ORACLE (contracts/protocol/LendPoolAddressesProvider.sol#30) is never used in LendPoolAddressesProvider (contracts/protocol/LendPool
AddressesProvider.sol#20-430)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

contracts/protocol/LendPoolConfigurator.sol

```
LendPoolConfigurator._checkReserveNoLiquidity(address) (contracts/protocol/LendPoolConfigurator.sol#675-681) uses a dangerous strict equality:
        - require(bool,string)(availableLiquidity == 0 && reserveData.currentLiquidityRate == 0,Errors.LPC_RESERVE_LIQUIDITY_NOT_0) (contracts/protocol/LendPoolConfigu
rator.sol#680)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

LendPoolConfigurator.setReserveInterestRateAddress(address[],address) (contracts/protocol/LendPoolConfigurator.sol#209-220) has external calls inside a loop: cachedPoo
l.setReserveInterestRateAddress(assets[i],rateAddress) (contracts/protocol/LendPoolConfigurator.sol#213)
LendPoolConfigurator.batchConfigReserve(ILendPoolConfigurator.ConfigReserveInput[]) (contracts/protocol/LendPoolConfigurator.sol#226-242) has external calls inside a l
oop: currentConfig = cachedPool.getReserveConfiguration(inputs[i].asset) (contracts/protocol/LendPoolConfigurator.sol#230)
LendPoolConfigurator.batchConfigReserve(ILendPoolConfigurator.ConfigReserveInput[]) (contracts/protocol/LendPoolConfigurator.sol#226-242) has external calls inside a l
oop: cachedPool.setReserveConfiguration(inputs[i].asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#234)
LendPoolConfigurator.setActiveFlagOnNftByTokenId(address[],uint256[],bool) (contracts/protocol/LendPoolConfigurator.sol#272-298) has external calls inside a loop: curr
entConfig = cachedPool.getNftConfigByTokenId(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#283)
LendPoolConfigurator.setActiveFlagOnNftByTokenId(address[],uint256[],bool) (contracts/protocol/LendPoolConfigurator.sol#272-298) has external calls inside a loop: cach
edPool.setNftConfigByTokenId(assets[i],tokenIds[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#286)
LendPoolConfigurator.setFreezeFlagOnNftByTokenId(address[],uint256[],bool) (contracts/protocol/LendPoolConfigurator.sol#323-346) has external calls inside a loop: curr
entConfig = cachedPool.getNftConfigByTokenId(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#331)
LendPoolConfigurator.setFreezeFlagOnNftByTokenId(address[],uint256[],bool) (contracts/protocol/LendPoolConfigurator.sol#323-346) has external calls inside a loop: cach
edPool.setNftConfigByTokenId(assets[i],tokenIds[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#334)
LendPoolConfigurator.setNftMaxSupplyAndTokenId(address[],uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#489-505) has external calls inside a loop: cache
dPool.setNftMaxSupplyAndTokenId(assets[i],maxSupply,maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#497)
LendPoolConfigurator.batchConfigNft(ILendPoolConfigurator.ConfigNftInput[]) (contracts/protocol/LendPoolConfigurator.sol#512-576) has external calls inside a loop: cur
rentConfig = cachedPool.getNftConfigByTokenId(inputs[i].asset,inputs[i].tokenId) (contracts/protocol/LendPoolConfigurator.sol#516-519)
LendPoolConfigurator.batchConfigNft(ILendPoolConfigurator.ConfigNftInput[]) (contracts/protocol/LendPoolConfigurator.sol#512-576) has external calls inside a loop: cac
hedPool.setNftConfigByTokenId(inputs[i].asset,inputs[i].tokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#549)
LendPoolConfigurator.batchConfigNft(ILendPoolConfigurator.ConfigNftInput[]) (contracts/protocol/LendPoolConfigurator.sol#512-576) has external calls inside a loop: cac
hedPool.setNftMaxSupplyAndTokenId(inputs[i].asset,inputs[i].maxSupply,inputs[i].maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#569)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in LendPoolConfigurator.batchConfigNft(ILendPoolConfigurator.ConfigNftInput[]) (contracts/protocol/LendPoolConfigurator.sol#512-576):
        External calls:
        - cachedPool.setNftConfigByTokenId(inputs[i].asset,inputs[i].tokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#549)
        Event emitted after the call(s):
        - NftAuctionChanged(inputs[i].asset,inputs[i].tokenId,inputs[i].redeemDuration,inputs[i].auctionDuration,inputs[i].redeemFine) (contracts/protocol/LendPoolConf
igurator.sol#558-564)
        - NftConfigurationChanged(inputs[i].asset,inputs[i].tokenId,inputs[i].baseLTV,inputs[i].liquidationThreshold,inputs[i].liquidationBonus) (contracts/protocol/Le
ndPoolConfigurator.sol#551-557)
        - NftMinBidFineChanged(inputs[i].asset,inputs[i].tokenId,inputs[i].minBidFine) (contracts/protocol/LendPoolConfigurator.sol#566)
        - NftRedeemThresholdChanged(inputs[i].asset,inputs[i].tokenId,inputs[i].redeemThreshold) (contracts/protocol/LendPoolConfigurator.sol#565)
Reentrancy in LendPoolConfigurator.batchConfigNft(ILendPoolConfigurator.ConfigNftInput[]) (contracts/protocol/LendPoolConfigurator.sol#512-576):
        External calls:
        - cachedPool.setNftConfigByTokenId(inputs[i].asset,inputs[i].tokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#549)
        - cachedPool.setNftMaxSupplyAndTokenId(inputs[i].asset,inputs[i].maxSupply,inputs[i].maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#569)
        Event emitted after the call(s):
        - NftMaxSupplyAndTokenIdChanged(inputs[i].asset,inputs[i].maxSupply,inputs[i].maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#570)
Reentrancy in LendPoolConfigurator.batchConfigReserve(ILendPoolConfigurator.ConfigReserveInput[]) (contracts/protocol/LendPoolConfigurator.sol#226-242):
        External calls:
        - cachedPool.setReserveConfiguration(inputs[i].asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#234)
        Event emitted after the call(s):
        - ReserveFactorChanged(inputs[i].asset,inputs[i].reserveFactor) (contracts/protocol/LendPoolConfigurator.sol#236)
Reentrancy in LendPoolConfigurator.configureNftAsAuction(address,uint256,uint256,uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#418-439):
        External calls:
        - cachedPool.setNftConfigByTokenId(asset,nftTokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#436)
        Event emitted after the call(s):
        - NftAuctionChanged(asset,nftTokenId,redeemDuration,auctionDuration,redeemFine) (contracts/protocol/LendPoolConfigurator.sol#438)
Reentrancy in LendPoolConfigurator.configureNftAsCollateral(address,uint256,uint256,uint256,uint256,uint256,uint256,uint256,bool,bool) (contracts/protocol/Lend
PoolConfigurator.sol#360-409):
        External calls:
        - cachedPool.setNftConfigByTokenId(asset,nftTokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#404)
        - INFTOracle(_addressesProvider.getNFTOracle()).setNFTPrice(asset,nftTokenId,newPrice) (contracts/protocol/LendPoolConfigurator.sol#406)
        Event emitted after the call(s):
        - NftConfigurationChanged(asset,nftTokenId,ltv,liquidationThreshold,liquidationBonus) (contracts/protocol/LendPoolConfigurator.sol#408)
Reentrancy in LendPoolConfigurator.setActiveFlagOnNft(address,bool) (contracts/protocol/LendPoolConfigurator.sol#249-264):
        External calls:
        - cachedPool.setNftConfiguration(asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#257)
        Event emitted after the call(s):
        - NftActivated(asset) (contracts/protocol/LendPoolConfigurator.sol#260)
        - NftDeactivated(asset) (contracts/protocol/LendPoolConfigurator.sol#262)
Reentrancy in LendPoolConfigurator.setActiveFlagOnNftByTokenId(address[],uint256[],bool) (contracts/protocol/LendPoolConfigurator.sol#272-298):
        External calls:
        - cachedPool.setNftConfigByTokenId(assets[i],tokenIds[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#286)
        Event emitted after the call(s):
        - NftTokenActivated(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#289)
        - NftTokenDeactivated(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#291)
Reentrancy in LendPoolConfigurator.setActiveFlagOnReserve(address,bool) (contracts/protocol/LendPoolConfigurator.sol#153-167):
        External calls:
        - cachedPool.setReserveConfiguration(asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#161)
        Event emitted after the call(s):
        - ReserveActivated(asset) (contracts/protocol/LendPoolConfigurator.sol#163)
        - ReserveDeactivated(asset) (contracts/protocol/LendPoolConfigurator.sol#165)
Reentrancy in LendPoolConfigurator.setBorrowingFlagOnReserve(address,bool) (contracts/protocol/LendPoolConfigurator.sol#129-146):
        External calls:
        - cachedPool.setReserveConfiguration(asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#139)
        Event emitted after the call(s):
        - BorrowingDisabledOnReserve(asset) (contracts/protocol/LendPoolConfigurator.sol#144)
        - BorrowingEnabledOnReserve(asset) (contracts/protocol/LendPoolConfigurator.sol#142)
Reentrancy in LendPoolConfigurator.setFreezeFlagOnNft(address,bool) (contracts/protocol/LendPoolConfigurator.sol#305-315):
        External calls:
        - cachedPool.setNftConfiguration(asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#309)
        Event emitted after the call(s):
        - NftFrozen(asset) (contracts/protocol/LendPoolConfigurator.sol#311)
        - NftUnfrozen(asset) (contracts/protocol/LendPoolConfigurator.sol#313)
Reentrancy in LendPoolConfigurator.setFreezeFlagOnNftByTokenId(address[],uint256[],bool) (contracts/protocol/LendPoolConfigurator.sol#323-346):
        External calls:
        - cachedPool.setNftConfigByTokenId(assets[i],tokenIds[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#334)
        Event emitted after the call(s):
        - NftTokenFrozen(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#337)
        - NftTokenUnfrozen(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#339)
Reentrancy in LendPoolConfigurator.setFreezeFlagOnReserve(address,bool) (contracts/protocol/LendPoolConfigurator.sol#174-186):
        External calls:
        - cachedPool.setReserveConfiguration(asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#179)
        Event emitted after the call(s):
        - ReserveFrozen(asset) (contracts/protocol/LendPoolConfigurator.sol#182)
        - ReserveUnfrozen(asset) (contracts/protocol/LendPoolConfigurator.sol#184)
```

```
Reentrancy in LendPoolConfigurator.setNftMaxSupplyAndTokenId(address[],uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#489-505):
        External calls:
        - cachedPool.setNftMaxSupplyAndTokenId(assets[i],maxSupply,maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#497)
        Event emitted after the call(s):
        - NftMaxSupplyAndTokenIdChanged(assets[i],maxSupply,maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#499)
Reentrancy in LendPoolConfigurator.setNftMinBidFine(address,uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#468-481):
        External calls:
        - cachedPool.setNftConfigByTokenId(asset,nftTokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#478)
        Event emitted after the call(s):
        - NftMinBidFineChanged(asset,nftTokenId,minBidFine) (contracts/protocol/LendPoolConfigurator.sol#480)
Reentrancy in LendPoolConfigurator.setNftRedeemThreshold(address,uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#447-460):
        External calls:
        - cachedPool.setNftConfigByTokenId(asset,nftTokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#457)
        Event emitted after the call(s):
        - NftRedeemThresholdChanged(asset,nftTokenId,redeemThreshold) (contracts/protocol/LendPoolConfigurator.sol#459)
Reentrancy in LendPoolConfigurator.setReserveFactor(address,uint256) (contracts/protocol/LendPoolConfigurator.sol#193-202):
        External calls:
        - cachedPool.setReserveConfiguration(asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#199)
        Event emitted after the call(s):
        - ReserveFactorChanged(asset,reserveFactor) (contracts/protocol/LendPoolConfigurator.sol#201)
Reentrancy in LendPoolConfigurator.setReserveInterestRateAddress(address[],address) (contracts/protocol/LendPoolConfigurator.sol#209-220):
        External calls:
        - cachedPool.setReserveInterestRateAddress(assets[i],rateAddress) (contracts/protocol/LendPoolConfigurator.sol#213)
        Event emitted after the call(s):
        - ReserveInterestRateChanged(assets[i],rateAddress) (contracts/protocol/LendPoolConfigurator.sol#214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

contracts/protocol/LendPoolLoan.sol

```
NFTXHelper.sellNFTX(ILendPoolAddressesProvider,address,uint256,address) (contracts/libraries/nftx/NFTXHelper.sol#31-84) ignores return value by nftxVault.mint(tokenId
s,new uint256[](1)) (contracts/libraries/nftx/NFTXHelper.sol#56)
NFTXHelper.sellNFTX(ILendPoolAddressesProvider,address,uint256,address) (contracts/libraries/nftx/NFTXHelper.sol#31-84) ignores return value by IERC20Upgradeable(vaul
tAddress).approve(sushiSwapRouterAddress,depositAmount) (contracts/libraries/nftx/NFTXHelper.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Reentrancy in LendPoolLoan.createLoan(address,address,address,uint256,address,address,uint256,uint256) (contracts/protocol/LendPoolLoan.sol#65-108):
        External calls:
        - IERC721Upgradeable(nftAsset).safeTransferFrom(_msgSender(),address(this),nftTokenId) (contracts/protocol/LendPoolLoan.sol#87)
        - IUNFT(uNftAddress).mint(onBehalfOf,nftTokenId) (contracts/protocol/LendPoolLoan.sol#89)
        State variables written after the call(s):
        - loanData.loanId = loanId (contracts/protocol/LendPoolLoan.sol#93)
        - loanData.state = DataTypes.LoanState.Active (contracts/protocol/LendPoolLoan.sol#94)
        - loanData.borrower = onBehalfOf (contracts/protocol/LendPoolLoan.sol#95)
        - loanData.nftAsset = nftAsset (contracts/protocol/LendPoolLoan.sol#96)
        - loanData.nftTokenId = nftTokenId (contracts/protocol/LendPoolLoan.sol#97)
        - loanData.reserveAsset = reserveAsset (contracts/protocol/LendPoolLoan.sol#98)
        - loanData.scaledAmount = amountScaled (contracts/protocol/LendPoolLoan.sol#99)
        - _nftTotalCollateral[nftAsset] += 1 (contracts/protocol/LendPoolLoan.sol#103)
        - _userNftCollateral[onBehalfOf][nftAsset] += 1 (contracts/protocol/LendPoolLoan.sol#101)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in LendPoolLoan.createLoan(address,address,address,uint256,address,address,uint256,uint256) (contracts/protocol/LendPoolLoan.sol#65-108):
        External calls:
        - IERC721Upgradeable(nftAsset).safeTransferFrom(_msgSender(),address(this),nftTokenId) (contracts/protocol/LendPoolLoan.sol#87)
        - IUNFT(uNftAddress).mint(onBehalfOf,nftTokenId) (contracts/protocol/LendPoolLoan.sol#89)
        Event emitted after the call(s):
        - LoanCreated(initiator,onBehalfOf,loanId,nftAsset,nftTokenId,reserveAsset,amount,borrowIndex) (contracts/protocol/LendPoolLoan.sol#105)
Reentrancy in LendPoolLoan.liquidateLoan(address,uint256,address,uint256,uint256) (contracts/protocol/LendPoolLoan.sol#272-312):
        External calls:
        - IUNFT(uNftAddress).burn(loan.nftTokenId) (contracts/protocol/LendPoolLoan.sol#299)
        - IERC721Upgradeable(loan.nftAsset).safeTransferFrom(address(this),_msgSender(),loan.nftTokenId) (contracts/protocol/LendPoolLoan.sol#301)
        Event emitted after the call(s):
        - LoanLiquidated(initiator,loanId,loan.nftAsset,loan.nftTokenId,loan.reserveAsset,borrowAmount,borrowIndex) (contracts/protocol/LendPoolLoan.sol#303-311)
Reentrancy in LendPoolLoan.liquidateLoanNFTX(uint256,address,uint256,uint256) (contracts/protocol/LendPoolLoan.sol#317-359):
        External calls:
        - IUNFT(uNftAddress).burn(loan.nftTokenId) (contracts/protocol/LendPoolLoan.sol#342)
        - sellPrice = NFTXHelper.sellNFTX(_addressesProvider,loan.nftAsset,loan.nftTokenId,loan.reserveAsset) (contracts/protocol/LendPoolLoan.sol#348)
        Event emitted after the call(s):
        - LoanLiquidatedNFTX(loanId,loan.nftAsset,loan.nftTokenId,loan.reserveAsset,borrowAmount,borrowIndex,sellPrice) (contracts/protocol/LendPoolLoan.sol#350-358)
Reentrancy in LendPoolLoan.repayLoan(address,uint256,address,uint256,uint256) (contracts/protocol/LendPoolLoan.sol#158-189):
        External calls:
        - IUNFT(uNftAddress).burn(loan.nftTokenId) (contracts/protocol/LendPoolLoan.sol#184)
        - IERC721Upgradeable(loan.nftAsset).safeTransferFrom(address(this),_msgSender(),loan.nftTokenId) (contracts/protocol/LendPoolLoan.sol#186)
        Event emitted after the call(s):
        - LoanRepaid(initiator,loanId,loan.nftAsset,loan.nftTokenId,loan.reserveAsset,amount,borrowIndex) (contracts/protocol/LendPoolLoan.sol#188)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

LendPoolLoan.auctionLoan(address,uint256,address,uint256,uint256,uint256) (contracts/protocol/LendPoolLoan.sol#194-236) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(loan.state == DataTypes.LoanState.Active,Errors.LPL_INVALID_LOAN_STATE) (contracts/protocol/LendPoolLoan.sol#209)
        - require(bool,string)(loan.state == DataTypes.LoanState.Auction,Errors.LPL_INVALID_LOAN_STATE) (contracts/protocol/LendPoolLoan.sol#215)
        - require(bool,string)(bidPrice > loan.bidPrice,Errors.LPL_BID_PRICE_LESS_THAN_HIGHEST_PRICE) (contracts/protocol/LendPoolLoan.sol#217)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

NFTXHelper.getNFTXPrice(ILendPoolAddressesProvider,address,uint256,address) (contracts/libraries/nftx/NFTXHelper.sol#92-136) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Redundant expression "operator (contracts/protocol/LendPoolLoan.sol#367)" inLendPoolLoan (contracts/protocol/LendPoolLoan.sol#22-468)
Redundant expression "from (contracts/protocol/LendPoolLoan.sol#368)" inLendPoolLoan (contracts/protocol/LendPoolLoan.sol#22-468)
Redundant expression "tokenId (contracts/protocol/LendPoolLoan.sol#369)" inLendPoolLoan (contracts/protocol/LendPoolLoan.sol#22-468)
Redundant expression "data (contracts/protocol/LendPoolLoan.sol#370)" inLendPoolLoan (contracts/protocol/LendPoolLoan.sol#22-468)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

contracts/protocol/NFTOracle.sol

```
NFTOracle.getMultipleNFTPrices(address[],uint256[]) (contracts/protocol/NFTOracle.sol#233-252) has external calls inside a loop: _nftPrices[i] = this.getNFTPrice(_coll
ections[i],_tokenIds[i]) (contracts/protocol/NFTOracle.sol#245)
NFTOracle.getNFTPriceNFTX(address,uint256) (contracts/protocol/NFTOracle.sol#270-303) has external calls inside a loop: nftxVault.allValidNFTs(tokenIds) (contracts/pro
tocol/NFTOracle.sol#286)
NFTOracle.getNFTPriceNFTX(address,uint256) (contracts/protocol/NFTOracle.sol#270-303) has external calls inside a loop: swapPath[1] = IUniswapV2Router02(sushiswapRoute
r).WETH() (contracts/protocol/NFTOracle.sol#290)
NFTOracle.getNFTPriceNFTX(address,uint256) (contracts/protocol/NFTOracle.sol#270-303) has external calls inside a loop: amountIn = 1 ** IERC20MetadataUpgradeable(addre
ss(nftxVault)).decimals() (contracts/protocol/NFTOracle.sol#293)
NFTOracle.getNFTPriceNFTX(address,uint256) (contracts/protocol/NFTOracle.sol#270-303) has external calls inside a loop: amounts = IUniswapV2Router02(sushiswapRouter).g
etAmountsOut(amountIn,swapPath) (contracts/protocol/NFTOracle.sol#294)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
```

contracts/protocol/PunkGateway.sol

```
PunkGateway._repayETH(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#365-396) sends eth to arbitrary user
        Dangerous calls:
        - (paybackAmount,burn) = _wethGateway.repayETH{value: repayDebtAmount}(address(wrappedPunks),punkIndex,amount) (contracts/protocol/PunkGateway.sol#385-389)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

EmergencyTokenRecoveryUpgradeable.emergencyERC20Transfer(address,address,uint256) (contracts/protocol/EmergencyTokenRecoveryUpgradeable.sol#29-35) ignores return value
 by IERC20Upgradeable(token).transfer(to,amount) (contracts/protocol/EmergencyTokenRecoveryUpgradeable.sol#34)
PunkGateway.borrow(address,uint256,uint256,address,uint16) (contracts/protocol/PunkGateway.sol#174-189) ignores return value by IERC20Upgradeable(reserveAsset).transfe
r(onBehalfOf,amount) (contracts/protocol/PunkGateway.sol#188)
PunkGateway._repay(uint256,uint256) (contracts/protocol/PunkGateway.sol#215-239) ignores return value by IERC20Upgradeable(reserve).transferFrom(msg.sender,address(thi
s),amount) (contracts/protocol/PunkGateway.sol#229)
PunkGateway.auction(uint256,uint256,address) (contracts/protocol/PunkGateway.sol#241-259) ignores return value by IERC20Upgradeable(reserve).transferFrom(msg.sender,ad
dress(this),bidPrice) (contracts/protocol/PunkGateway.sol#256)
PunkGateway.redeem(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#261-283) ignores return value by IERC20Upgradeable(loan.reserveAsset).transferFrom(msg.
sender,address(this),(amount + bidFine)) (contracts/protocol/PunkGateway.sol#274)
PunkGateway.liquidate(uint256,uint256) (contracts/protocol/PunkGateway.sol#285-308) ignores return value by IERC20Upgradeable(loan.reserveAsset).transferFrom(msg.sende
r,address(this),amount) (contracts/protocol/PunkGateway.sol#296)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

PunkGateway.authorizeLendPoolERC20(address[]) (contracts/protocol/PunkGateway.sol#102-112) ignores return value by IERC20Upgradeable(tokens[i]).approve(address(_getLen
dPool()),0) (contracts/protocol/PunkGateway.sol#105)
PunkGateway.authorizeLendPoolERC20(address[]) (contracts/protocol/PunkGateway.sol#102-112) ignores return value by IERC20Upgradeable(tokens[i]).approve(address(_getLen
dPool()),type()(uint256).max) (contracts/protocol/PunkGateway.sol#106)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

EmergencyTokenRecoveryUpgradeable.emergencyEtherTransfer(address,uint256).to (contracts/protocol/EmergencyTokenRecoveryUpgradeable.sol#72) lacks a zero-check on :
        - (success) = to.call{value: amount}(new bytes(0)) (contracts/protocol/EmergencyTokenRecoveryUpgradeable.sol#73)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

PunkGateway._getLendPool() (contracts/protocol/PunkGateway.sol#87-89) has external calls inside a loop: ILendPool(_addressProvider.getLendPool()) (contracts/protocol/P
unkGateway.sol#88)
PunkGateway.authorizeLendPoolERC20(address[]) (contracts/protocol/PunkGateway.sol#102-112) has external calls inside a loop: IERC20Upgradeable(tokens[i]).approve(addre
ss(_getLendPool()),0) (contracts/protocol/PunkGateway.sol#105)
PunkGateway.authorizeLendPoolERC20(address[]) (contracts/protocol/PunkGateway.sol#102-112) has external calls inside a loop: IERC20Upgradeable(tokens[i]).approve(addre
ss(_getLendPool()),type()(uint256).max) (contracts/protocol/PunkGateway.sol#106)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in PunkGateway.initialize(address,address,address,address) (contracts/protocol/PunkGateway.sol#63-82):
        External calls:
        - wrappedPunks.registerProxy() (contracts/protocol/PunkGateway.sol#77)
        - proxy = wrappedPunks.proxyInfo(address(this)) (contracts/protocol/PunkGateway.sol#78)
        State variables written after the call(s):
        - proxy = wrappedPunks.proxyInfo(address(this)) (contracts/protocol/PunkGateway.sol#78)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in EmergencyTokenRecoveryUpgradeable.emergencyEtherTransfer(address,uint256) (contracts/protocol/EmergencyTokenRecoveryUpgradeable.sol#72-76):
        External calls:
        - (success) = to.call{value: amount}(new bytes(0)) (contracts/protocol/EmergencyTokenRecoveryUpgradeable.sol#73)
        Event emitted after the call(s):
        - EmergencyEtherTransfer(to,amount) (contracts/protocol/EmergencyTokenRecoveryUpgradeable.sol#75)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

PunkGateway._checkValidCallerAndOnBehalfOf(address) (contracts/protocol/PunkGateway.sol#142-147) compares to a boolean constant:
        -require(bool,string)((onBehalfOf == _msgSender()) || (_callerWhitelists[_msgSender()] == true),Errors.CALLER_NOT_ONBEHALFOF_OR_IN_WHITELIST) (contracts/protoc
ol/PunkGateway.sol#143-146)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

contracts/protocol/WETHGateway.sol

```
WETHGateway._repayETH(address,uint256,uint256,uint256) (contracts/protocol/WETHGateway.sol#214-239) sends eth to arbitrary user
        Dangerous calls:
        - WETH.deposit{value: repayDebtAmount}() (contracts/protocol/WETHGateway.sol#235)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

WETHGateway.withdrawETH(uint256,address) (contracts/protocol/WETHGateway.sol#144-162) ignores return value by uWETH.transferFrom(msg.sender,address(this),amountToWithd
raw) (contracts/protocol/WETHGateway.sol#158)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

WETHGateway.initialize(address,address) (contracts/protocol/WETHGateway.sol#54-63) ignores return value by WETH.approve(address(_getLendPool()),type()(uint256).max) (c
ontracts/protocol/WETHGateway.sol#62)
WETHGateway.withdrawETH(uint256,address) (contracts/protocol/WETHGateway.sol#144-162) ignores return value by cachedPool.withdraw(address(WETH),amountToWithdraw,addres
s(this)) (contracts/protocol/WETHGateway.sol#159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

WETHGateway._getLendPool() (contracts/protocol/WETHGateway.sol#68-70) has external calls inside a loop: ILendPool(_addressProvider.getLendPool()) (contracts/protocol/W
ETHGateway.sol#69)
WETHGateway.authorizeLendPoolNFT(address[]) (contracts/protocol/WETHGateway.sol#83-92) has external calls inside a loop: IERC721Upgradeable(nftAssets[i]).setApprovalFo
rAll(address(_getLendPool()),true) (contracts/protocol/WETHGateway.sol#86)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

WETHGateway._checkValidCallerAndOnBehalfOf(address) (contracts/protocol/WETHGateway.sol#122-127) compares to a boolean constant:
        -require(bool,string)((onBehalfOf == _msgSender()) || (_callerWhitelists[_msgSender()] == true),Errors.CALLER_NOT_ONBEHALFOF_OR_IN_WHITELIST) (contracts/protoc
ol/WETHGateway.sol#123-126)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Low level call in WETHGateway._safeTransferETH(address,uint256) (contracts/protocol/WETHGateway.sol#337-340):
        - (success) = to.call{value: value}(new bytes(0)) (contracts/protocol/WETHGateway.sol#338)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

- No major issues were found by Slither.
- The sends eth to arbitrary user findings are false positives.
- The ignores return value by IERC20Upgradeable findings are false positives because the contract uses the SafeERC20Upgradeable library.
- All the reentrancy vulnerabilities were checked individually, and they are all false positives.

4.2 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:

contracts/protocol/LendPool.sol

Report for contracts/protocol/LendPool.sol
https://dashboard.mythx.io/#/console/analyses/d123824c-17af-4f0e-93c9-4bc7ac7948fe

Line	SWC Title	Severity	Short Description
49	(SWC-123) Requirement Violation	Low	Requirement violation.
86	(SWC-107) Reentrancy	Low	Write to persistent state following external call
386	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
442	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
628	(SWC-123) Requirement Violation	Low	Requirement violation.

contracts/protocol/LendPoolAddressesProvider.sol

Report for contracts/protocol/LendPoolAddressesProvider.sol
https://dashboard.mythx.io/#/console/analyses/f8362f4f-6cb2-46ab-9c0c-838065539f01

Line	SWC Title	Severity	Short Description
19	(SWC-123) Requirement Violation	Low	Requirement violation.
79	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
121	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
144	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
235	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
387	(SWC-123) Requirement Violation	Low	Requirement violation.
406	(SWC-123) Requirement Violation	Low	Requirement violation.
408	(SWC-107) Reentrancy	Low	Read of persistent state following external call.

contracts/protocol/NFTOracle.sol

Report for contracts/protocol/NFTOracle.sol
https://dashboard.mythx.io/#/console/analyses/9aaaccac-7762-4ac9-b015-4820a19dbee2

Line	SWC Title	Severity	Short Description
13	(SWC-123) Requirement Violation	Low	Requirement violation.
259	(SWC-123) Requirement Violation	Low	Requirement violation.

contracts/protocol/LendPoolLoan.sol

Report for contracts/protocol/LendPoolLoan.sol
https://dashboard.mythx.io/#/console/analyses/091a3d96-cab1-477e-81d3-afb2f36de81c

Line	SWC Title	Severity	Short Description
22	(SWC-123) Requirement Violation	Low	Requirement violation.
57	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
57	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
486	(SWC-123) Requirement Violation	Low	Requirement violation.

- No major issues were discovered by MythX.
- The requirement violation findings are all false positives.
- Writing to the persistent state following external call findings are all false positives.

THANK YOU FOR CHOOSING

// HALBORN