HALBORN

Unlockd Finance - Sudoswap and Yearn Integration

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 2nd, 2023 - February 10th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/09/2023	Manuel Garcia
0.2	Document Updates	01/10/2023	Manuel Garcia
0.3	Final Draft	02/10/2023	Manuel Garcia
0.4	Draft Review	02/10/2023	Roberto Reigada
0.5	Draft Review	02/10/2023	Piotr Cielas
0.6	Draft Review	02/10/2023	Gabi Urrutia
1.0	Remediation Plan	03/20/2023	Manuel García
1.1	Remediation Plan Review	03/20/2023	Ataberk Yavuzer
1.2	Remediation Plan Review	03/20/2023	Piotr Cielas
1.3	Remediation Plan Review	03/20/2023	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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com
Manuel Garcia	Halborn	Manuel.Diaz@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Unlockd Finance is a Decentralized Finance protocol that unlocks democratized NFT liquidity allowing DeFi users to compound their wealth with a NFT-backed, cross-chain, instant loan maintaining 100% of holder perks.

Unlockd Finance engaged Halborn to conduct a security audit on their smart contracts beginning on January 2nd, 2023 and ending on February 10th, 2023 . The security assessment was scoped to the smart contracts provided in the unlockd GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

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:

- 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 multiple risks, which have been 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 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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

Code repositories:

1. Sudoswap integration:

- Repository: unlockd
- Commit ID: 9b218a11704ca5fc46454c25b6e99232c2e07df6
- Smart contracts in scope:
    1. contracts/protocol/LendPool.sol
        1.1. liquidateSudoSwap()
    2. contracts/protocol/LendPoolLoan.sol
        2.1. liquidateLoanSudoSwap()
    3. contracts/libraries/logic/LiquidateMarketsLogic.sol
        3.1. executeLiquidateSudoSwap()
    4. contracts/libraries/logic/ValidationLogic.sol
        4.1. validateLiquidateMarkets()
    5. contracts/libraries/logic/GenericLogic.sol
        5.1 calculateLoanLiquidatePrice()
    6. contracts/libraries/markets/SudoSwapSeller.sol

2. Yearn integration:

- Repository: unlockd
- Commit ID: 60400feb8de409261bdbca6979dd21c639537717
- Smart contracts in scope:
    1. contracts/protocol/UToken.sol
        1.1. depositReserves()
        1.2. withdrawReserves()
        1.3. getAvailableLiquidity()
    2. contracts/libraries/logic/LendingLogic.sol
        2.1. executeDepositYearn()
        2.2. executeWithdrawYearn()
        2.3. calculateYearnAvailableLiquidityInReserve()

3. Remmediations:
    - Commit ID: unlockd

Out-of-scope:

- third-party libraries and dependencies
- economic attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	1	4

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - LACK OF SLIPPAGE CONTROL	High	SOLVED - 02/06/2023
HAL-02 - ERC20 TOKENS FEE ON TRANSFER CAN LEAD TO UNDERCOLLATERALIZATION	Low	RISK ACCEPTED
HAL-03 - NFTS CAN BE LIQUIDATED BEFORE THEY ARE PUT UP AT AUCTION	Informational	ACKNOWLEDGED
HAL-04 - MISSING PRICE FUNCTION FOR SUDOSWAP ON DATA PROVIDER	Informational	SOLVED - 03/20/2023
HAL-05 - >0 CONSUMES MORE GAS THAN !=0 FOR UINTS	Informational	ACKNOWLEDGED
HAL-06 - MISLEADING DOCUMENTATION IN SOME OF THE FUNCTIONS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF SLIPPAGE CONTROL - HIGH

Description:

The market liquidation functionality allows the LendPoolLiquidator to liquidate NFTs used as collateral in markets such as NFTx and Sudoswap. Furthermore, the liquidate functions also uses Uniswap pools to exchange WETH to the underlying asset of the reserve, so the loan can be re-payed after liquidating the collateral.

Exchange routers such as Uniswap and Sudoswap usually require the user to specify a amountOutMin parameter to ensure that the amount of tokens they receive is greater or equal to amountOutMin. This parameter is set to 0 for both Uniswap and Sudoswap routers.

By setting this parameter to 0, the NFT for Sudoswap and the token for Uniswap can be swapped at any price. As Sudoswap pairs tend to have low liquidity, a malicious user could manipulate the pool by performing a sandwich attack. For a more detailed explanation, please see the Proof of Concept section below.

Code Location:

Listing 1: src/libraries/markets/SudoSwapSeller.sol (Line 45)

```
27 function sellSudoSwap(
28   ILendPoolAddressesProvider addressesProvider,
29   address nftAsset,
30   uint256 nftTokenId,
31   address LSSVMPair
32 ) internal returns (uint256 amount) {
33   address LSSVMRouterAddress = addressesProvider.getLSSVMRouter();
34   address lendPoolAddress = addressesProvider.getLendPool();
35
36   ILSSVMRouter LSSVMRouter = ILSSVMRouter(LSSVMRouterAddress);
37
38   uint256[] memory nftTokenIds = new uint256[](1);
```

```
39   nftTokenIds[0] = nftTokenId;
40
41   PairSwapSpecific[] memory pairSwaps = new PairSwapSpecific[](1);
42   pairSwaps[0] = PairSwapSpecific({pair: ILSSVMPair(LSSVMPair),
↳ nftIds: nftTokenIds});
43
44   IERC721Upgradeable(nftAsset).approve(LSSVMRouterAddress,
↳ nftTokenId);
45   amount = LSSVMRouter.swapNFTsForToken(pairSwaps, 0,
↳ lendPoolAddress, block.timestamp);
46 }
```

Listing 2: src/libraries/markets/NFTXSeller.sol (Line 72)

```
60 address[] memory swapPath;
61 if (reserveAsset != address(WETH)) {
62   swapPath = new address[](3);
63   swapPath[2] = reserveAsset;
64 } else {
65   swapPath = new address[](2);
66 }
67 swapPath[0] = vaultAddress;
68 swapPath[1] = WETH;
69
70 uint256[] memory amounts = IUniswapV2Router02(
↳ sushiSwapRouterAddress).swapExactTokensForTokens(
71   depositAmount,
72   0,
73   swapPath,
74   lendPoolAddress,
75   block.timestamp
76 );
77
```

Listing 3: src/libraries/logic/LiquidateMarketsLogic.sol (Line 320)

```
312 if (loanData.reserveAsset == WETH) {
313   IWETH(WETH).deposit{value: priceSudoSwap}();
314 } else {
315   address[] memory swapPath = new address[](2);
316   swapPath[0] = WETH;
317   swapPath[1] = loanData.reserveAsset;
318
```

```
319   uint256[] memory amounts = IUniswapV2Router02(
 ↳ sushiSwapRouterAddress).swapExactETHForTokens{value: priceSudoSwap
 ↳ }(
320     0,
321     swapPath,
322     address(this),
323     block.timestamp
324   );
325   priceSudoSwap = amounts[1];
326 }
```

Proof of Concept:

1. The liquidator calls the liquidateSudoSwap() function to liquidate an NFT asset.
2. A bot monitoring the mempool frontruns the transaction, providing liquidity to the Sudoswap pair and decreasing NFT value.
3. liquidateSudoSwap() transaction gets executed at a much lower price than expected.
4. As the underlying obtained from the TX is not enough to cover the borrowed position, tokens are withdrawn from the treasury. Therefore, the treasury is drained, but the transaction does not revert.
5. The bot buys the liquidity he provided earlier, and he can now get the NFT at a much lower price, making profit from the transaction.

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

It is recommended to adjust slippage accordingly and set a value in the amountOutMin parameter.

Remediation Plan:

SOLVED: The Unlockd Finance team solved this issue on commit 60400feb8de409261bdbca6979dd21c639537717: the amountOutMin parameter is now set by the liquidator when calling the function.

3.2 (HAL-02) ERC20 TOKENS FEE ON TRANSFER CAN LEAD TO UNDERCOLLATERALIZATION - LOW

Description:

The Unlockd protocol can use any ERC20 token set by the pool administrator as a reserve for lending and borrowing. However, it is important to note that some tokens may charge fees on transfers, meaning for every transfer of the token contract retains a small fee. However, the protocol relies on the specified amount by the user to calculate collateralization.

If the protocol ever uses an ERC20 token with fee on transfer enabled as a reserve, or one of the used tokens used as reserve enables this functionality, the protocol may suffer from under collateralization.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to calculate the amount of underlying tokens by calculating the delta of the balance before and after the token transfer, this way the calculated amount is correct even if fee on transfer is enabled. Moreover, some routers like Uniswap have dedicated functions for these use cases.

Remediation Plan:

RISK ACCEPTED: The Unlockd Finance team accepted the risk after confirming that they will only use WETH, which does not implement transfer fee or deflationary functionalities.

3.3 (HAL-03) NFTS CAN BE LIQUIDATED BEFORE THEY ARE PUT UP AT AUCTION - INFORMATIONAL

Description:

In the Unlockd protocol, when the value of the NFT used as collateral decreases below a certain threshold, the position becomes unhealthy. Usually, the NFT is put up at auction first and if there are not any bids, it can then be liquidated in one of the markets. A loan can have 6 possible states, although for the market liquidation only 3 are relevant for this issue:

- Active: the loan has been initialized, funds have been delivered to the borrower and the collateral is held.
- Auction: the loan is put up for auction and the highest bid can claim it.
- Defaulted: the loan was delinquent and collateral was claimed by the liquidator. This is a terminal state.

The market liquidation is triggered when the auction time has passed and there were no bids. The loan then enters the active state; however, loans are also in active state even before the auction, so it is technically possible to liquidate a loan in the market accidentally even before it undergoes the auction process as the `validateLiquidateMarkets` function in the `ValidationLogic` library checks for the loan to be in active state.

Listing 4: src/libraries/logic/ValidationLogic.sol (Line 310)

```
288 function validateLiquidateMarkets(
289   DataTypes.ReserveData storage reserveData,
290   DataTypes.NftData storage nftData,
291   DataTypes.NftConfigurationMap storage nftConfig,
292   DataTypes.LoanData memory loanData
293 ) internal view {
294   require(nftData.uNftAddress != address(0), Errors.
 ↳ LPC_INVALID_UNFT_ADDRESS);
```

```
295   require(reserveData.uTokenAddress != address(0), Errors.
 ↳ VL_INVALID_RESERVE_ADDRESS);
296
297   require(reserveData.configuration.getActive(), Errors.
 ↳ VL_NO_ACTIVE_RESERVE);
298
299   require(nftData.configuration.getActive(), Errors.
 ↳ VL_NO_ACTIVE_NFT);
300
301   /**
302    * @dev additional check for individual asset
303    */
304   require(nftConfig.getActive(), Errors.VL_NO_ACTIVE_NFT);
305
306   /**
307    * @dev Loan requires to be in `Active` state. The Markets
 ↳ liquidate process is triggered if there has not been any auction
308    * and the auction time has passed. In that case, loan is not in
 ↳  `Auction` nor `Defaulted`,  and needs to be liquidated in a third
 ↳ -party market.
309    */
310   require(loanData.state == DataTypes.LoanState.Active, Errors.
 ↳ LPL_INVALID_LOAN_STATE);
311 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to create an additional state to reflect a loan can be liquidated in a market once the auction has taken place to avoid liquidating it accidentally.

Remediation Plan:

ACKNOWLEDGED: The Unlockd Finance team acknowledged this issue as this is the intended behavior, as even if the NFT can be liquidated without going through the auction process, the health factor must be less than 1.

3.4 (HAL-04) MISSING PRICE FUNCTION FOR SUDOSWAP ON DATA PROVIDER - INFORMATIONAL

Description:

The Unlockd Protocol appears to have the UnlockdProtocolDataProvider contract to provide data about the protocol, presumably for off-chain purposes. As shown in the snippet below, there is a function to obtain the price of a NFT before liquidation on the NFTx market.

Listing 5: src/misc/UnlockdProtocolDataProvider.sol

```
326 /* CAUTION: Price uint is ETH based (WEI, 18 decimals) */
327 /**
328 @dev returns the NFT price for a given NFT valued by NFTX
329 @param asset the NFT collection
330 @param tokenId the NFT token Id
331  */
332 function getNFTXPrice(address asset, uint256 tokenId, address
 ↳ reserveAsset) external view returns (uint256) {
333   return NFTXSeller.getNFTXPrice(ADDRESSES_PROVIDER, asset,
 ↳ tokenId, reserveAsset);
334 }
```

However, there is no function to return the estimated price before liquidating the NFT at the Sudoswap market. Which might be useful to calculate the minimum amount the liquidator is willing to receive in order to liquidate the collateral.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Add a helper function as the one for NFTx for the Sudoswap market to the data provider.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue by adding the getStartingBidPrice function, which retrieves the price from the Sudoswap pair on the commit ID: e09df2aef1fab73381455205c6a3ea193a63eca0.

3.5 (HAL-05) >0 CONSUMES MORE GAS THAN !=0 FOR UINTS - INFORMATIONAL

Description:

The use of `>` consumes more gas than `!=`. There are some cases where both can be used indistinctly, such as in unsigned integers where numbers can't be negative, and as such, there is only a need to check that a number is not 0.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use `!=` instead of `>` in cases where both can be used, such as uints.

Remediation Plan:

ACKNOWLEDGED: The Unlockd Finance team acknowledged the issue.

3.6 (HAL-06) MISLEADING DOCUMENTATION IN SOME OF THE FUNCTIONS - INFORMATIONAL

Description:

Some functions had misleading documentation corresponding to other functions. Although this has no impact on security, it is important to ensure the protocol is properly documented for auditors and developers that may want to build on top of it.

Risk Level:

Likelihood - 1
Impact - 1

Code Location:

Listing 6: src/libraries/logic/LendingLogic.sol

```
82 /**
83    * @notice Implements the yearn vault withdraw feature. Through
↳ `executeWithdrawYearn()`, users withdrawn assets are withdrawn
↳ from the YVault.
84    * @dev Emits the `WithdrawYearn()` event.
85    * @param addressesProvider The protocol current address
↳ provider
86    * @return availableLiquidityInReserve The available liquidity
↳ in reserve format
87    */
88   function calculateYearnAvailableLiquidityInReserve(
```

Recommendation:

Ensure every function has proper documentation relative to the function.

Remediation Plan:

ACKNOWLEDGED: The Unlockd Finance team acknowledged the issue.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the 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:

LendPool.sol

```
LendPool.rescue(IERC20,address,uint256,bool) (src/protocol/LendPool.sol#1046-1058) sends eth to arbitrary user
        Dangerous calls:
        - (sent) = to.call{value: amount}() (src/protocol/LendPool.sol#1053)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

LendPool.getNftLiquidatePrice(address,uint256).vars (src/protocol/LendPool.sol#702) 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 (src/protocol/LendPool.sol#1048) lacks a zero-check on :
                - (sent) = to.call{value: amount}() (src/protocol/LendPool.sol#1053)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in LendPool.setTreasuryAddress(address,address) (src/protocol/LendPool.sol#1115-1119):
        External calls:
        - IUToken(uToken).setTreasuryAddress(treasury) (src/protocol/LendPool.sol#1117)
        Event emitted after the call(s):
        - TreasuryAddressUpdated(uToken,treasury) (src/protocol/LendPool.sol#1118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Pragma version0.8.4 (src/interfaces/ICryptoPunksMarket.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IDebtToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IIncentivesController.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IInterestRate.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPool.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolLoan.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/INFTOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IReserveOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IScaledBalanceToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IUToken.sol#2) allows old versions
Pragma version>=0.6.2 (src/interfaces/IUniswapV2Router01.sol#3) allows old versions
Pragma version>=0.6.2 (src/interfaces/IUniswapV2Router02.sol#3) allows old versions
Pragma version0.8.4 (src/interfaces/IWETH.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/nftx/INFTXVault.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/nftx/INFTXVaultFactoryV2.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/sudoswap/ILSSVMPair.sol#2) allows old versions
Pragma version0.8.4 (src/protocol/LendPool.sol#2) allows old versions
Pragma version0.8.4 (src/protocol/LendPoolStorage.sol#2) allows old versions
```

LendPoolLoan.sol

AUTOMATED TESTING

```
Reentrancy in LendPoolLoan.createLoan(address,address,address,uint256,address,address,uint256,uint256) (src/protocol/LendPoolLoan.sol#64-107):
        External calls:
        - IERC721Upgradeable(nftAsset).safeTransferFrom(_msgSender(),address(this),nftTokenId) (src/protocol/LendPoolLoan.sol#86)
        - IUNFT(uNftAddress).mint(onBehalfOf,nftTokenId) (src/protocol/LendPoolLoan.sol#88)
        State variables written after the call(s):
        - loanData.loanId = loanId (src/protocol/LendPoolLoan.sol#92)
        - loanData.state = DataTypes.LoanState.Active (src/protocol/LendPoolLoan.sol#93)
        - loanData.borrower = onBehalfOf (src/protocol/LendPoolLoan.sol#94)
        - loanData.nftAsset = nftAsset (src/protocol/LendPoolLoan.sol#95)
        - loanData.nftTokenId = nftTokenId (src/protocol/LendPoolLoan.sol#96)
        - loanData.reserveAsset = reserveAsset (src/protocol/LendPoolLoan.sol#97)
        - loanData.scaledAmount = amountScaled (src/protocol/LendPoolLoan.sol#98)
        - _nftTotalCollateral[nftAsset] += 1 (src/protocol/LendPoolLoan.sol#102)
        - _userNftCollateral[onBehalfOf][nftAsset] += 1 (src/protocol/LendPoolLoan.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in LendPoolLoan.createLoan(address,address,address,uint256,address,address,uint256,uint256) (src/protocol/LendPoolLoan.sol#64-107):
        External calls:
        - IERC721Upgradeable(nftAsset).safeTransferFrom(_msgSender(),address(this),nftTokenId) (src/protocol/LendPoolLoan.sol#86)
        - IUNFT(uNftAddress).mint(onBehalfOf,nftTokenId) (src/protocol/LendPoolLoan.sol#88)
        Event emitted after the call(s):
        - LoanCreated(initiator,onBehalfOf,loanId,nftAsset,nftTokenId,reserveAsset,amount,borrowIndex) (src/protocol/LendPoolLoan.sol#104)
Reentrancy in LendPoolLoan.liquidateLoan(address,uint256,address,uint256,uint256) (src/protocol/LendPoolLoan.sol#271-311):
        External calls:
        - IUNFT(uNftAddress).burn(loan.nftTokenId) (src/protocol/LendPoolLoan.sol#298)
        - IERC721Upgradeable(loan.nftAsset).safeTransferFrom(address(this),_msgSender(),loan.nftTokenId) (src/protocol/LendPoolLoan.sol#300)
        Event emitted after the call(s):
        - LoanLiquidated(initiator,loanId,loan.nftAsset,loan.nftTokenId,loan.reserveAsset,borrowAmount,borrowIndex) (src/protocol/LendPoolLoan.sol#302-310)
Reentrancy in LendPoolLoan.liquidateLoanNFTX(uint256,address,uint256,uint256,uint256) (src/protocol/LendPoolLoan.sol#316-364):
        External calls:
        - IUNFT(uNftAddress).burn(loan.nftTokenId) (src/protocol/LendPoolLoan.sol#342)
        - sellPrice = NFTXSeller.sellNFTX(_addressesProvider,loan.nftAsset,loan.nftTokenId,loan.reserveAsset,amountOutMin) (src/protocol/LendPoolLoan.sol#347-353)
        Event emitted after the call(s):
        - LoanLiquidatedNFTX(loanId,loan.nftAsset,loan.nftTokenId,loan.reserveAsset,borrowAmount,borrowIndex,sellPrice) (src/protocol/LendPoolLoan.sol#355-363)
Reentrancy in LendPoolLoan.liquidateLoanSudoSwap(uint256,address,uint256,uint256,DataTypes.SudoSwapParams) (src/protocol/LendPoolLoan.sol#369-418):
        External calls:
        - IUNFT(uNftAddress).burn(loan.nftTokenId) (src/protocol/LendPoolLoan.sol#395)
        - sellPrice = SudoSwapSeller.sellSudoSwap(_addressesProvider,loan.nftAsset,loan.nftTokenId,sudoswapParams.LSSVMPair,sudoswapParams.amountOutMinSudoswap) (src/pr
otocol/LendPoolLoan.sol#400-406)
        Event emitted after the call(s):
        - LoanLiquidatedSudoSwap(loanId,loan.nftAsset,loan.nftTokenId,loan.reserveAsset,borrowAmount,borrowIndex,sellPrice,sudoswapParams.LSSVMPair) (src/protocol/LendP
oolLoan.sol#408-417)
Reentrancy in LendPoolLoan.repayLoan(address,uint256,address,uint256,uint256) (src/protocol/LendPoolLoan.sol#157-188):
        External calls:
        - IUNFT(uNftAddress).burn(loan.nftTokenId) (src/protocol/LendPoolLoan.sol#183)
        - IERC721Upgradeable(loan.nftAsset).safeTransferFrom(address(this),_msgSender(),loan.nftTokenId) (src/protocol/LendPoolLoan.sol#185)
        Event emitted after the call(s):
        - LoanRepaid(initiator,loanId,loan.nftAsset,loan.nftTokenId,loan.reserveAsset,amount,borrowIndex) (src/protocol/LendPoolLoan.sol#187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

LendPoolLoan.auctionLoan(address,uint256,address,uint256,uint256,uint256) (src/protocol/LendPoolLoan.sol#193-235) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(loan.state == DataTypes.LoanState.Active,Errors.LPL_INVALID_LOAN_STATE) (src/protocol/LendPoolLoan.sol#208)
        - require(bool,string)(loan.state == DataTypes.LoanState.Auction,Errors.LPL_INVALID_LOAN_STATE) (src/protocol/LendPoolLoan.sol#214)
        - require(bool,string)(bidPrice > loan.bidPrice,Errors.LPL_BID_PRICE_LESS_THAN_HIGHEST_PRICE) (src/protocol/LendPoolLoan.sol#216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Pragma version0.8.4 (src/interfaces/ILendPool.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolLoan.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IUNFT.sol#2) allows old versions
Pragma version>=0.6.2 (src/interfaces/IUniswapV2Router01.sol#3) allows old versions
Pragma version>=0.6.2 (src/interfaces/IUniswapV2Router02.sol#3) allows old versions
Pragma version0.8.4 (src/interfaces/nftx/INFTXVault.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/nftx/INFTXVaultFactoryV2.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/sudoswap/ILSSVMPair.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/sudoswap/ILSSVMRouter.sol#2) allows old versions
Pragma version0.8.4 (src/protocol/LendPoolLoan.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

Parameter ILendPool.liquidateSudoSwap(address,uint256,uint256,address,uint256).LSSVMPair (src/interfaces/ILendPool.sol#342) is not in mixedCase
Function IUniswapV2Router01.WETH() (src/interfaces/IUniswapV2Router01.sol#8) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

LiquidateMarketsLogic.sol

```
Pragma version0.8.4 (src/interfaces/IDebtToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IIncentivesController.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IInterestRate.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPool.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolLoan.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/INFTOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IReserveOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IScaledBalanceToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IUToken.sol#2) allows old versions
Pragma version>=0.6.2 (src/interfaces/IUniswapV2Router01.sol#3) allows old versions
Pragma version>=0.6.2 (src/interfaces/IUniswapV2Router02.sol#3) allows old versions
Pragma version0.8.4 (src/interfaces/IWETH.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

Parameter ILendPool.liquidateSudoSwap(address,uint256,uint256,address,uint256).LSSVMPair (src/interfaces/ILendPool.sol#342) is not in mixedCase
Function IUToken.UNDERLYING_ASSET_ADDRESS() (src/interfaces/IUToken.sol#136) is not in mixedCase
Function IUToken.RESERVE_TREASURY_ADDRESS() (src/interfaces/IUToken.sol#141) is not in mixedCase
Function IUniswapV2Router01.WETH() (src/interfaces/IUniswapV2Router01.sol#8) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (src/interfaces/IUniswapV2Router01.sol#13) is t
oo similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (src/interfaces/IUniswapV2Router01.sol#14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
./src/libraries/logic/LiquidateMarketsLogic.sol analyzed (33 contracts with 84 detectors), 19 result(s) found
```

ValidationLogic.sol

```
Pragma version0.8.4 (src/interfaces/IDebtToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IIncentivesController.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IInterestRate.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPool.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolLoan.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/INFTOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IReserveOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IScaledBalanceToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IUToken.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

Parameter ILendPool.liquidateSudoSwap(address,uint256,uint256,address,uint256).LSSVMPair (src/interfaces/ILendPool.sol#342) is not in mixedCase
Function IUToken.UNDERLYING_ASSET_ADDRESS() (src/interfaces/IUToken.sol#136) is not in mixedCase
Function IUToken.RESERVE_TREASURY_ADDRESS() (src/interfaces/IUToken.sol#141) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
./src/libraries/logic/ValidationLogic.sol analyzed (29 contracts with 84 detectors), 14 result(s) found
```

GenericLogic.sol

```
Pragma version0.8.4 (src/interfaces/IDebtToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IIncentivesController.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IInterestRate.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolLoan.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/INFTOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IReserveOracleGetter.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IScaledBalanceToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IUToken.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

Function IUToken.UNDERLYING_ASSET_ADDRESS() (src/interfaces/IUToken.sol#136) is not in mixedCase
Function IUToken.RESERVE_TREASURY_ADDRESS() (src/interfaces/IUToken.sol#141) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
./src/libraries/logic/GenericLogic.sol analyzed (24 contracts with 84 detectors), 12 result(s) found
```

LendingLogic.sol

```
Pragma version0.8.4 (src/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/yearn/IYVault.sol#3) 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
./src/libraries/logic/LendingLogic.sol analyzed (10 contracts with 84 detectors), 3 result(s) found
```

UToken.sol

```
UToken.initialize(ILendPoolAddressesProvider,address,address,uint8,string,string).treasury (src/protocol/UToken.sol#52) lacks a zero-check on :
                - _treasury = treasury (src/protocol/UToken.sol#60)
UToken.initialize(ILendPoolAddressesProvider,address,address,uint8,string,string).underlyingAsset (src/protocol/UToken.sol#53) lacks a zero-check on :
                - _underlyingAsset = underlyingAsset (src/protocol/UToken.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in UToken._transfer(address,address,uint256,bool) (src/protocol/UToken.sol#301-317):
        External calls:
        - super._transfer(from,to,amount.rayDiv(index)) (src/protocol/UToken.sol#310)
                - _getIncentivesController().handleAction(sender,currentTotalSupply,oldSenderBalance) (src/protocol/IncentivizedERC20.sol#57)
                - _getIncentivesController().handleAction(recipient,currentTotalSupply,oldRecipientBalance) (src/protocol/IncentivizedERC20.sol#59)
        Event emitted after the call(s):
        - BalanceTransfer(from,to,amount,index) (src/protocol/UToken.sol#316)
Reentrancy in UToken.burn(address,address,uint256,uint256) (src/protocol/UToken.sol#81-95):
        External calls:
        - _burn(user,amountScaled) (src/protocol/UToken.sol#90)
                - _getIncentivesController().handleAction(account,oldTotalSupply,oldAccountBalance) (src/protocol/IncentivizedERC20.sol#82)
        - IERC20Upgradeable(_underlyingAsset).safeTransfer(receiverOfUnderlying,amount) (src/protocol/UToken.sol#92)
        Event emitted after the call(s):
        - Burn(user,receiverOfUnderlying,amount,index) (src/protocol/UToken.sol#94)
Reentrancy in UToken.mint(address,uint256,uint256) (src/protocol/UToken.sol#105-117):
        External calls:
        - _mint(user,amountScaled) (src/protocol/UToken.sol#112)
                - _getIncentivesController().handleAction(account,oldTotalSupply,oldAccountBalance) (src/protocol/IncentivizedERC20.sol#71)
        Event emitted after the call(s):
        - Mint(user,amount,index) (src/protocol/UToken.sol#114)
Reentrancy in UToken.mintToTreasury(uint256,uint256) (src/protocol/UToken.sol#148-163):
        External calls:
        - _mint(treasury,amount.rayDiv(index)) (src/protocol/UToken.sol#159)
                - _getIncentivesController().handleAction(account,oldTotalSupply,oldAccountBalance) (src/protocol/IncentivizedERC20.sol#71)
        Event emitted after the call(s):
        - Mint(treasury,amount,index) (src/protocol/UToken.sol#162)
        - Transfer(address(0),treasury,amount) (src/protocol/UToken.sol#161)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

UToken._getLendPoolConfigurator() (src/protocol/UToken.sol#289-291) is never used and should be removed
UToken._getUnderlyingAssetAddress() (src/protocol/UToken.sol#262-264) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version0.8.4 (src/interfaces/IIncentivesController.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPool.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/ILendPoolConfigurator.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IScaledBalanceToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/IUToken.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/yearn/IYVault.sol#3) allows old versions
Pragma version0.8.4 (src/protocol/IncentivizedERC20.sol#2) allows old versions
Pragma version0.8.4 (src/protocol/UToken.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

Parameter ILendPool.liquidateSudoSwap(address,uint256,uint256,address,uint256).LSSVMPair (src/interfaces/ILendPool.sol#342) is not in mixedCase
Function IUToken.UNDERLYING_ASSET_ADDRESS() (src/interfaces/IUToken.sol#136) is not in mixedCase
Function IUToken.RESERVE_TREASURY_ADDRESS() (src/interfaces/IUToken.sol#141) is not in mixedCase
Function IncentivizedERC20.__IncentivizedERC20_init(string,string,uint8) (src/protocol/IncentivizedERC20.sol#20-28) is not in mixedCase
Variable IncentivizedERC20.__gap (src/protocol/IncentivizedERC20.sol#86) is not in mixedCase
Function UToken.RESERVE_TREASURY_ADDRESS() (src/protocol/UToken.sol#237-239) is not in mixedCase
Function UToken.UNDERLYING_ASSET_ADDRESS() (src/protocol/UToken.sol#244-246) is not in mixedCase
Function UToken.POOL() (src/protocol/UToken.sol#251-253) is not in mixedCase
Variable UToken._addressProvider (src/protocol/UToken.sol#30) is not in mixedCase
Variable UToken._treasury (src/protocol/UToken.sol#31) is not in mixedCase
Variable UToken._underlyingAsset (src/protocol/UToken.sol#32) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
./src/protocol/UToken.sol analyzed (24 contracts with 84 detectors), 29 result(s) found
```

SudoSwapSeller.sol

```
Pragma version0.8.4 (src/interfaces/ILendPoolAddressesProvider.sol#2) allows old versions
Pragma version>=0.6.2 (src/interfaces/IUniswapV2Router01.sol#3) allows old versions
Pragma version>=0.6.2 (src/interfaces/IUniswapV2Router02.sol#3) allows old versions
Pragma version0.8.4 (src/interfaces/sudoswap/ILSSVMPair.sol#2) allows old versions
Pragma version0.8.4 (src/interfaces/sudoswap/ILSSVMRouter.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

Function IUniswapV2Router01.WETH() (src/interfaces/IUniswapV2Router01.sol#8) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (src/interfaces/IUniswapV2Router01.sol#13) is t
oo similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (src/interfaces/IUniswapV2Router01.sol#14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
./src/libraries/markets/SudoSwapSeller.sol analyzed (9 contracts with 84 detectors), 8 result(s) found
```

THANK YOU FOR CHOOSING

// HALBORN