Unlockd Finance - Protocol V2

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **October 26th, 2023 - December 11th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	11/20/2023
0.2	Document Update	12/13/2023
0.3	Draft Review	12/13/2023
0.4	Draft Review	12/13/2023
1.0	Remediation Plan	03/04/2024
1.1	Remediation Plan Review	03/04/2024
1.2	Remediation Plan Review	03/04/2024

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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

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

Unlockd Finance engaged Halborn to conduct a security assessment on their smart contracts beginning on October 26th, 2023 and ending on December 11th, 2023. The security assessment was scoped to the smart contracts provided in the UnlockdFinance/unlockd-v2 GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided 5 weeks for the engagement and assigned one full-time security engineer to verify the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, that were successfully addressed by Unlockd Finance. The main ones were the following:

- Review the functions of the modules and move all parameters that should not be user-controlled inside the Unlockd signed data structure parameter.
- Fix the loan health calculation to return the actual value based on the user's collateral.
- Review the functions of the modules and ensure that only authorized users can call them. If the function is to be called by arbitrary users, the information should be included in the function documentation.
- Fix the finalize and forceSell functions to only delete the loans that have no more assets remaining in them.
- Update the loan ID correctly in the finalize() function.

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 assessment. 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 assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walk-through.
- 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.
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (Foundry, Brownie).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker’s control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Unlockd Finance Protocol V2:

- Repository: UnlockdFinance/unlockd-v2
- Initial commit ID: 781ad1be131ae9c311e03f49b4e064315c18b39b
- Contract updates commit ID: 3c71181748383b4c58286a19cf943995295b0bc3
- Smart contracts in scope:
  - src/utils/BlockContext.sol
  - src/protocol/DebtToken.sol
  - src/protocol/UToken.sol
  - src/protocol/Unlockd.sol
  - src/protocol/adapters/ReservoirAdapter.sol
  - src/protocol/modules/Market.sol
  - src/protocol/modules/BuyNow.sol
  - src/protocol/modules/Installer.sol
  - src/protocol/modules/Auction.sol
  - src/protocol/modules/SellNow.sol
  - src/protocol/modules/Action.sol
  - src/protocol/modules/Manager.sol
  - src/libraries/utils/ReentrancyGuard.sol
  - src/libraries/utils/EIP712.sol
  - src/libraries/utils/tokens/ERC20Upgradeable.sol
  - src/libraries/signatures/ActionSign.sol
  - src/libraries/signatures/BuyNowSign.sol
  - src/libraries/signatures/MarketSign.sol
  - src/libraries/signatures/AuctionSign.sol
  - src/libraries/signatures/SellNowSign.sol
  - src/libraries/oracles/ReserveOracle.sol
  - src/libraries/helpers/Errors.sol
  - src/libraries/helpers/Constants.sol
  - src/libraries/math/PercentageMath.sol
  - src/libraries/math/MathUtils.sol
  - src/libraries/math/WadRayMath.sol

- src/libraries/proxy/UnlockdProxyAdmin.sol
- src/libraries/proxy/UnlockdMinimalProxy.sol
- src/libraries/proxy/UnlockdUpgradeableProxy.sol
- src/libraries/logic/GenericLogic.sol
- src/libraries/logic/OrderLogic.sol
- src/libraries/logic/SellNowLogic.sol
- src/libraries/logic/LoanLogic.sol
- src/libraries/logic/AssetLogic.sol
- src/libraries/logic/BuyNowLogic.sol
- src/libraries/logic/ValidationLogic.sol
- src/libraries/logic/ReserveLogic.sol
- src/libraries/storage/CoreStorage.sol
- src/libraries/storage/UTokenStorage.sol
- src/libraries/base/BaseSignature.sol
- src/libraries/base/InterestRate.sol
- src/libraries/base/BaseERC20.sol
- src/libraries/base/BaseCoreModule.sol
- src/libraries/base/BaseCore.sol
- src/libraries/configuration/ACLManager.sol
- src/types/DataTypes.sol
- src/deployer/DeployProtocol.sol
- src/deployer/DeployPeriphery.sol
- src/deployer/DeployUToken.sol
- src/deployer/DeployUTokenConfig.sol

Remediation Commit ID :

- 1e76e618

Out-of-scope :

- Third-party libraries and dependencies.

- Economic attacks.

- New features/implementations after/within the 3c71181 & 1e76e618 commit IDs.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	0	2	3	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) IMPROPER PARAMETER VALIDATION LEADS TO UNAUTHORIZED ACTIONS	Critical (10)	SOLVED - 01/25/2024
(HAL-02) IMPROPER COLLATERAL HEALTH CHECK LEADS TO BORROW FUNDS WITHOUT ENOUGH COLLATERAL	Critical (10)	SOLVED - 01/27/2024
(HAL-03) MISSING AUTHORIZATION CHECK IN CANCELCLAIM	Critical (10)	SOLVED - 01/25/2024
(HAL-04) LOAN IS DELETED INSTEAD OF ACTIVATION	Critical (10)	SOLVED - 12/19/2023
(HAL-05) LOANID IS NOT UPDATED PROPERLY DURING AUCTION FINALIZATION	Critical (10)	SOLVED - 03/04/2024
(HAL-06) LACK OF ORACLE DATA FEED VALIDATION	Medium (5.0)	SOLVED - 11/24/2023
(HAL-07) IMPROPER RESERVOIR WITHDRAWERC20 IMPLEMENTATION	Medium (5.0)	SOLVED - 12/19/2023
(HAL-08) RESERVE STATE IS NOT UPDATED IN CREATE	Low (2.5)	SOLVED - 01/19/2024
(HAL-09) DANGEROUS REENTRANT PATTERNS	Low (2.5)	SOLVED - 11/28/2023
(HAL-10) IMPROPER WITHDRAW LIQUIDITY CHECK	Low (2.5)	SOLVED - 12/19/2023
(HAL-11) MISSING ZERO ADDRESS CHECKS	Informational (0.8)	SOLVED - 11/28/2023
(HAL-12) MISSING EVENTS FOR CONTRACT OPERATIONS	Informational (0.8)	SOLVED - 01/16/2024

FINDINGS & TECH DETAILS

4.1 (HAL-01) IMPROPER PARAMETER VALIDATION LEADS TO UNAUTHORIZED ACTIONS - CRITICAL(10)

Description:

In the protocol, when calling functions, important configuration parameters (e.g., IDs and prices) are provided by Unlockd using a data structure verified by the sig signature parameter. However, it was identified that some important parameters are not included in this verified data structure and can be controlled by the caller.

For example, in the Action module, it has been identified that when a user takes out a loan, regardless of the SignAction loan configuration assigned to them, they can specify other assets to be locked instead of the ones that were supposed to be used as collateral. By exploiting this vulnerability, the user can borrow a loan and lock assets having significantly less value than the supposed collateral.

It was identified that this type of error occurs frequently in the contracts, and other modules are also affected. For example, the cancelClaim function in the Market module does not check whether the user-controlled orderId parameter matches the SignMarket data.

Code Location:

The uToken, amount and assets parameters of the borrow() function can be changed by the user:

Listing 1: src/protocol/modules/Action.sol (Lines 65-67)

```
64  function borrow(
65    address uToken,
66    uint256 amount,
67    DataTypes.Asset[] calldata assets,
68    DataTypes.SignAction calldata signAction,
69    DataTypes.EIP712Signature calldata sig
```

```
) external isUTokenAllowed(uToken) {
...
```

Listing 2: src/types/DataTypes.sol

```
struct Asset {
  address collection;
  uint256 tokenId;
}
```

However, the loan configuration is passed using the Unlockd controlled SignAction parameter:

Listing 3: src/types/DataTypes.sol

```
struct SignAction {
  SignLoanConfig loan;
  bytes32[] assets;
  uint256 nonce;
  uint256 deadline;
}
```

Listing 4: src/types/DataTypes.sol

```
struct SignLoanConfig {
  bytes32 loanId;
  uint256 aggLoanPrice;
  uint256 aggLtv;
  uint256 aggLiquidationThreshold;
  uint88 totalAssets;
  uint256 nonce;
  uint256 deadline;
}
```

Note that both the user-controlled and the signed parameters contain asset IDs in different formats.

The borrow() function, when taking a loan, only checks if the length of the user-controlled assets parameter is equal to totalAssets and use the token IDs from there:

Listing 5: src/protocol/modules/Action.sol (Lines 67,77,89-91)

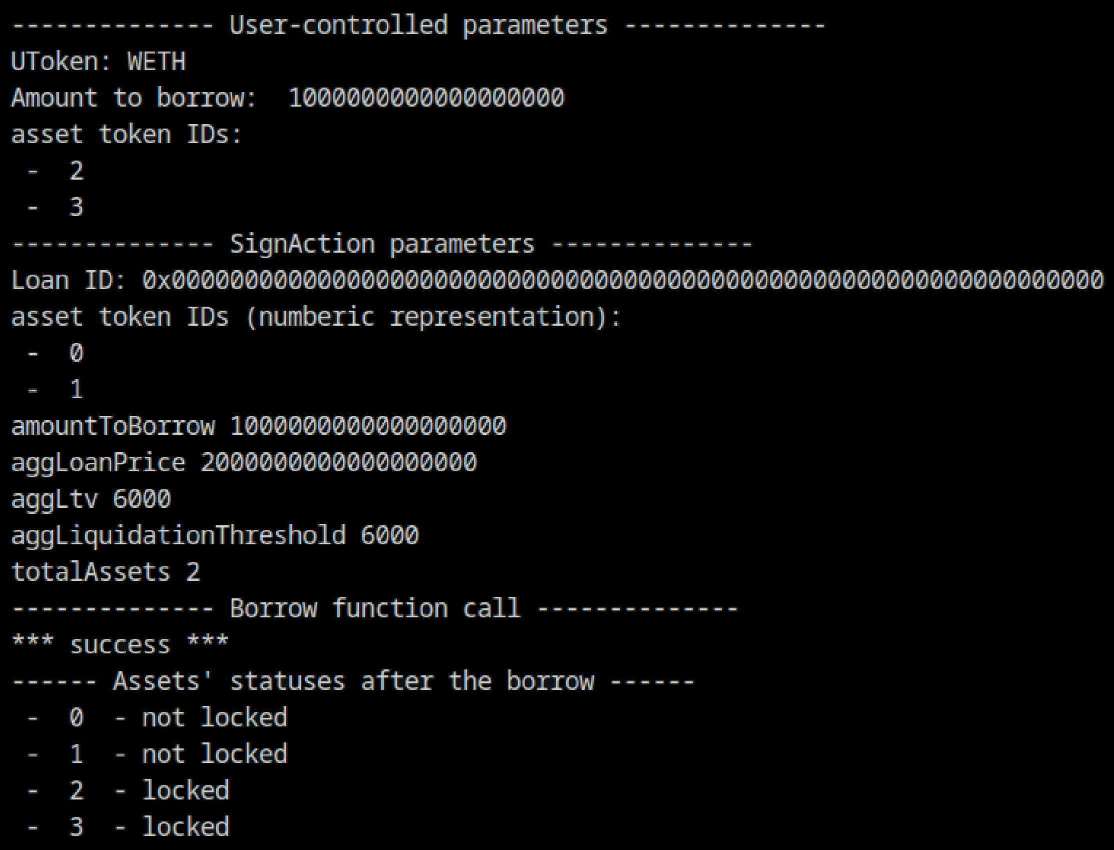
```
64   function borrow(
65     address uToken,
66     uint256 amount,
67     DataTypes.Asset[] calldata assets,
68     DataTypes.SignAction calldata signAction,
69     DataTypes.EIP712Signature calldata sig
70   ) external isUTokenAllowed(uToken) {
71     address msgSender = unpackTrailingParamMsgSender();
72     _checkHasUnlockdWallet(msgSender);
73
74     // We validate the signature
75     _validateSignature(msgSender, signAction, sig);
76
77     uint256 cachedAssets = assets.length;
78
79     DataTypes.ReserveData memory reserve = IUToken(uToken).
 ↳ getReserve();
80
81     // Generate the loanID
82     // Check if exist
83     DataTypes.Loan memory loan;
84     // New Loan
85     if (signAction.loan.loanId == 0) {
86       if (cachedAssets == 0) {
87         revert Errors.InvalidAssetAmount();
88       }
89       if (cachedAssets != signAction.loan.totalAssets) {
90         revert Errors.InvalidArrayLength();
91       }
92
93       // Create a new one
94       bytes32 loanId = LoanLogic.generateId(
95         msgSender,
96         signAction.loan.nonce,
97         signAction.loan.deadline
98       );
99       _loans[loanId].createLoan(
100        LoanLogic.ParamsCreateLoan({
101          msgSender: msgSender,
        uToken: uToken,
        underlyingAsset: reserve.underlyingAsset,
        // We added only when we lock the assets
        totalAssets: 0,
        loanId: loanId
      })
    );
    ...
}
```

Note that it was identified that the unit tests of the protocol generated the parameters of the functions using the same data source, making it difficult to build test cases where the token IDs provided by the user and Unlocked did not match. This increases the likelihood of not identifying similar vulnerabilities, as unit tests do not cover such cases.

Proof of Concept:

1. Create a signAction parameter for the user.
2. Call the borrow function using different assets as a parameter. The number of assets must be the same, and the user needs to be their owner.
3. Verify that the NFTs passed in the user-controlled parameter are locked instead of the ones passed in the signAction parameter.

The following image displays the used parameters and assets' statuses during the above borrowing process:



```
-------------- User-controlled parameters --------------
UToken: WETH
Amount to borrow:  1000000000000000000
asset token IDs:
 -  2
 -  3
-------------- SignAction parameters --------------
Loan ID: 0x0000000000000000000000000000000000000000000000000000000000000000
asset token IDs (numberic representation):
 -  0
 -  1
amountToBorrow 1000000000000000000
aggLoanPrice 2000000000000000000
aggLtv 6000
aggLiquidationThreshold 6000
totalAssets 2
-------------- Borrow function call --------------
*** success ***
------ Assets' statuses after the borrow ------
 -  0  - not locked
 -  1  - not locked
 -  2  - locked
 -  3  - locked
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

The functions of the modules should be reviewed, and all parameters that should not be user-controlled should be moved inside the Unlockd signed data structure parameters. This includes identifiers and other important parameters that determine the values inside the Unlockd signed data structure (e.g., leverage value depends on the used asset IDs).

It is also recommended to extend the unit tests with more customization options. This includes testing scenarios where the user-controlled parameters are generated independently of the signed parameters. For example, if the user is allowed to control the underlying asset of the loan, borrowing other underlying assets that WETH should be added to the tests.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commits 3f6e82a, 38cc3ef 4bcb910 by removing the uToken parameters and adding additional validation to the functions.

4.2 (HAL-02) IMPROPER COLLATERAL HEALTH CHECK LEADS TO BORROW FUNDS WITHOUT ENOUGH COLLATERAL - CRITICAL(10)

Description:

It was identified that the borrow() function in the Action module employs an inappropriate health factor check, and therefore, it is possible to borrow funds regardless of the collateral used for the loan.

Code Location:

The borrow amount is passed by the user:

Listing 6: src/protocol/modules/Action.sol

```
function borrow(
  address uToken,
  uint256 amount,
  DataTypes.Asset[] calldata assets,
  DataTypes.SignAction calldata signAction,
  DataTypes.EIP712Signature calldata sig
) external isUTokenAllowed(uToken) {
  address msgSender = unpackTrailingParamMsgSender();
  _checkHasUnlockdWallet(msgSender);
```

The validateFutureLoanState function is used to validate the state of the loan in the borrow() function:

Listing 7: src/protocol/modules/Action.sol

```
    // If the amount is 0 we don't need to borrow more
    if (amount != 0) {
      // We validate if the user can borrow
      ValidationLogic.validateFutureLoanState(
        ValidationLogic.ValidateLoanStateParams({
          user: msgSender,
          amount: amount,
          price: 0,
          reserveOracle: _reserveOracle,
          reserve: reserve,
          loanConfig: signAction.loan
        })
      );
```

The healthFactor calculation in the ValidationLogic library returns an invalid data, resulting in always passing the health check:

Listing 8: src/libraries/logic/ValidationLogic.sol

```
    (uint256 userCollateralBalance, uint256 userTotalDebt, uint256 healthFactor) = GenericLogic
      .calculateFutureLoanData(
        params.loanConfig.loanId,
        params.amount,
        params.price,
        params.user,
        params.reserveOracle,
        params.reserve,
        params.loanConfig
      );
```

The healthFactor is calculated based on the updatedDebt value.

Listing 9: src/libraries/logic/GenericLogic.sol

```
    // If the total assets are 0, then we need to calculate the collateral with the current value
    uint256 collateral = loanConfig.totalAssets == 0 ? price : loanConfig.aggLoanPrice;
    vars.totalCollateralInReserve = collateral.mulDiv(vars.reserveUnit, vars.reserveUnitPrice);

    uint256 updatedDebt = vars.totalDebtInReserve > amount ? vars.totalDebtInReserve - amount : 0;
    // Calculate the HF
    vars.healthFactor = calculateHealthFactorFromBalances(
      vars.totalCollateralInReserve,
      updatedDebt,
      loanConfig.aggLiquidationThreshold
    );
```

The functions return the maximum uint256 value as the totalDebt (updatedDebt) is 0:

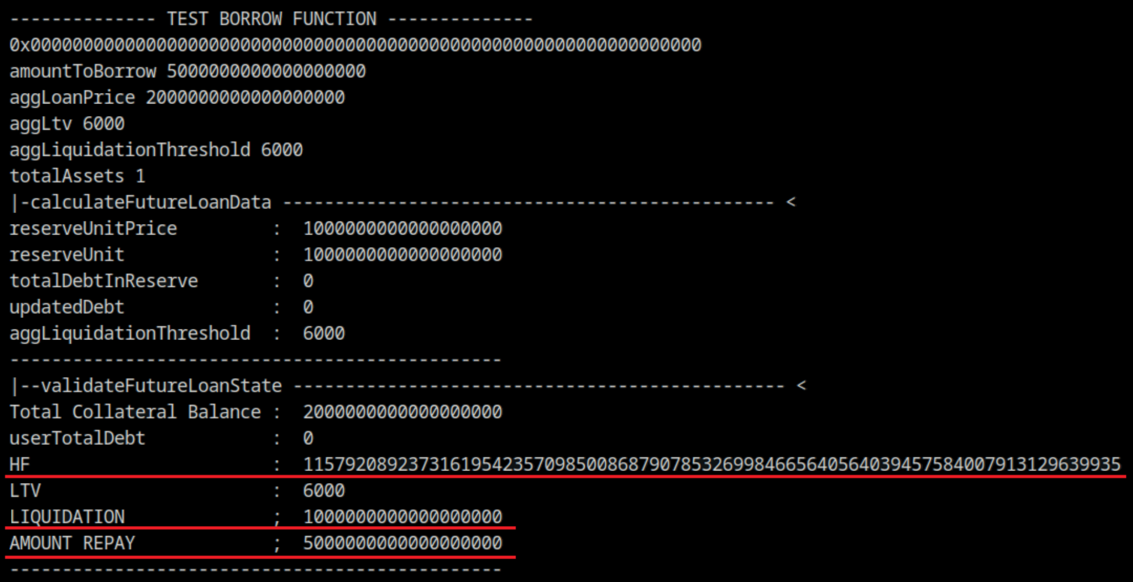
Listing 10: src/libraries/logic/GenericLogic.sol

```
function calculateHealthFactorFromBalances(
  uint256 totalCollateral,
  uint256 totalDebt,
  uint256 liquidationThreshold
) internal pure returns (uint256 healthFactor) {

  healthFactor = totalDebt == 0
    ? type(uint256).max
    : (totalCollateral.percentMul(liquidationThreshold)).wadDiv(totalDebt);

}
```

Proof of Concept:

1. Create a signAction parameter for the user.
2. Call the borrow function using a higher amount than the collateral.
3. Verify that the borrow action was successful.

The following image displays the state variables during the above borrowing process:



```
------------- TEST BORROW FUNCTION -------------
0x0000000000000000000000000000000000000000
amountToBorrow 5000000000000000000
aggLoanPrice 2000000000000000000
aggLtv 6000
aggLiquidationThreshold 6000
totalAssets 1
|-calculateFutureLoanData ---------------------------------------------- <
reserveUnitPrice          :  1000000000000000000
reserveUnit               :  1000000000000000000
totalDebtInReserve        :  0
updatedDebt               :  0
aggLiquidationThreshold   :  6000
----------------------------------------------
|--validateFutureLoanState ---------------------------------------------- <
Total Collateral Balance :  2000000000000000000
userTotalDebt             :  0
HF                        :  115792089237316195423570985008687907853269984665640564039457584007913129639935
LTV                       :  6000
LIQUIDATION               ;  1000000000000000000
AMOUNT REPAY              ;  5000000000000000000
----------------------------------------------
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

The loan health calculation should be fixed to return the actual value based on the user's collateral.

The unit tests of the modules should be extended with tests verifying that the health check is returning the expected value in every use case.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 92e8b46 by fixing the health factor calculation.

4.3 (HAL-03) MISSING AUTHORIZATION CHECK IN CANCELCLAIM - CRITICAL(10)

Description:

It was identified that the cancelClaim() function in the Market contract can be executed by anyone, not by just the order's winner. A user can get a signMarket data and then use it to cancel other users' claim by passing their orderId when calling the cancelClaim function. Canceling the claim will prevent the winner from getting their assets once an auction is ended.

Code Location:

The cancelClaim() function lacks authorization checks. The msgSender is only used to verify the nonce in the _validateSignature() function. If the msgSender has the same nonce, it will pass the verification check.

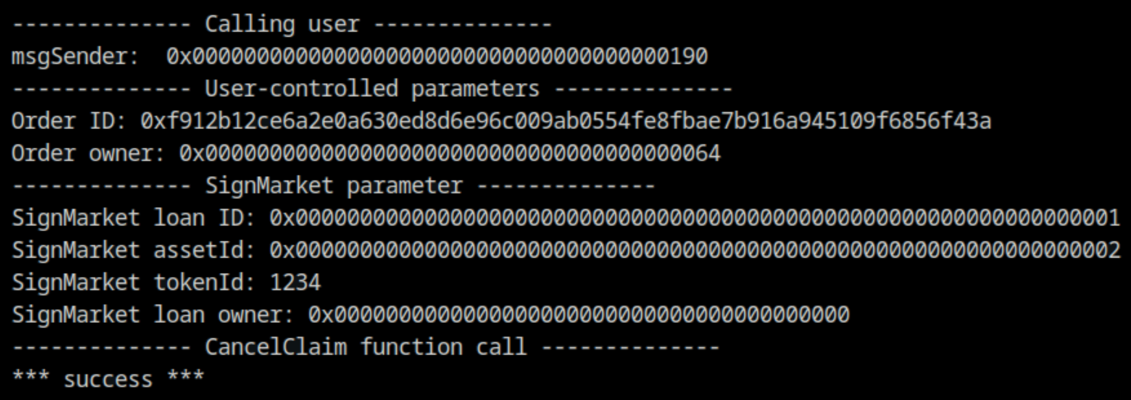
Listing 11: src/protocol/modules/Market.sol

```
function cancelClaim(
  bool claimOnUWallet,
  bytes32 orderId,
  DataTypes.SignMarket calldata signMarket,
  DataTypes.EIP712Signature calldata sig
) external {
  address msgSender = unpackTrailingParamMsgSender();
  _validateSignature(msgSender, signMarket, sig);
  DataTypes.Order memory order = _orders[orderId];

  // Get the loan asigned to the Order
  DataTypes.Loan storage loan = _loans[order.offer.loanId];

  {
    // Avoid stack too deep
    uint88 loanTotalAssets = loan.totalAssets;
    DataTypes.LoanState loanState = loan.state;
    // Validate if the order is ended
    ValidationLogic.validateOrderClaim(
      signMarket.loan.totalAssets,
      order,
      loanTotalAssets,
      loanState
    );
  }
```

Proof of Concept:

1. Win an auction with ACTOR1.
2. Create a SignMarket parameter for ACTOR2 with different data.
3. Call the cancelClaim function with ACTOR2 using their SignMarket parameter and cancel the order won by ACTOR1.
4. Verify that the order was canceled.

The following image displays the parameter variables during the above cancelClaim process:



```
 -------------- Calling user --------------
 msgSender:  0x0000000000000000000000000000000000000190
 -------------- User-controlled parameters --------------
 Order ID: 0xf912b12ce6a2e0a630ed8d6e96c009ab0554fe8fbae7b916a945109f6856f43a
 Order owner: 0x0000000000000000000000000000000000000064
 -------------- SignMarket parameter --------------
 SignMarket loan ID: 0x0000000000000000000000000000000000000000000000000000000000000001
 SignMarket assetId: 0x0000000000000000000000000000000000000000000000000000000000000002
 SignMarket tokenId: 1234
 SignMarket loan owner: 0x0000000000000000000000000000000000000000
 -------------- CancelClaim function call --------------
 *** success ***
```

Note that the msgSender and the SignMarket parameter are not related to the canceled order.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

The functions of modules should be reviewed to ensure that only authorized users can call them. If the function is to be called by arbitrary users, the information should be included in the function documentation.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit, 4bcb910 by adding additional validation to the cancelClaim function.

4.4 (HAL-04) LOAN IS DELETED INSTEAD OF ACTIVATION - CRITICAL(10)

Description:

It was identified that the finalize() function in the Auction module improperly deleted the still active offer loan instead of activating it because of an improper conditional check applied in the function.

It was also identified that the forceSell() function in the SellNow module has a similar issue, deleting the whole loan when selling only one asset.

Assets remaining in the deleted loans will be stuck, and their owners will not be able to retrieve them.

Code Location:

The finalize() and forceSell() functions improperly handle the usecase when the totalAssets of the loan is 1:

Listing 12: src/protocol/modules/Auction.sol

```
    if (signAuction.loan.totalAssets > 1) {
      // Activate loan
      loan.activate();
      loan.totalAssets = signAuction.loan.totalAssets;
    } else {
      // If there is only one we can remove the loan
      delete _loans[offerLoanId];
    }
```

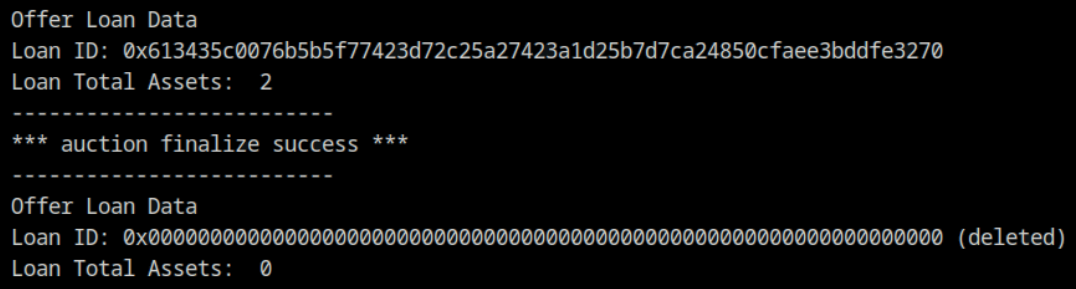
Listing 13: src/protocol/modules/SellNow.sol

```
    if (signSellNow.loan.totalAssets > 1) {
      // Activate loan
      _loans[loan.loanId].activate();
      _loans[signSellNow.loan.loanId].totalAssets = signSellNow.loan.totalAssets;
    } else {
      // If there is only one we can remove the loan
      delete _loans[loan.loanId];
    }
```

Proof of Concept:

1. Finalize a liquidation auction having two total assets.
2. Verify that instead of activation, the loan is deleted during finalization.

The following image displays the loan's state change during the above auction finalization process:



```
Offer Loan Data
Loan ID: 0x613435c0076b5b5f77423d72c25a27423a1d25b7d7ca24850cfaee3bddfe3270
Loan Total Assets:  2
--------------------------
*** auction finalize success ***
--------------------------
Offer Loan Data
Loan ID: 0x0000000000000000000000000000000000000000000000000000000000000000 (deleted)
Loan Total Assets:  0
```

Note that after auctioning 1 NFT, the loan with the remaining 1 asset was deleted.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

The conditions should be fixed so that the loans are only deleted when no more assets remain in them.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commits ce7b94f and 76a0374.

4.5 (HAL-05) LOANID IS NOT UPDATED PROPERLY DURING AUCTION FINALIZATION - CRITICAL(10)

Description:

It was identified that in the finalize() function in the Auction module, if the bidder has a loan with the new asset, the loanId is set to the offer's loanId instead of the bidder's loanId.

If the bidder has no loan, the loanId is not set to zero. Therefore, the assets won in the auction remained locked in their wallets.

Code Location:

The finalize() function is setting the loanId to loan.loanId instead of order.bid.loanId:

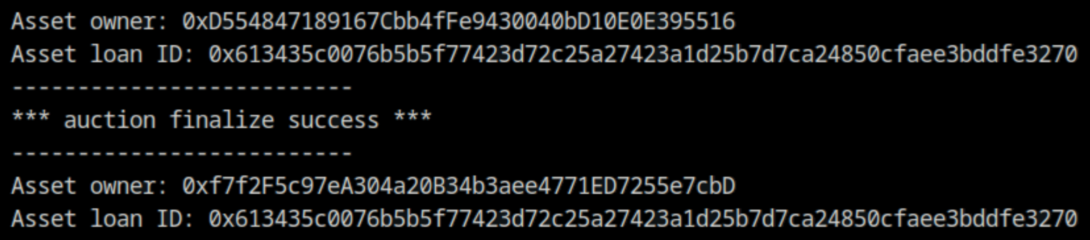
Listing 14: src/protocol/modules/Auction.sol (Line 441)

```
430     // If the bidder has a loan with the new asset
431     // we need to activate the loan and change the ownership to
  ↳ this new loan
432     if (order.bid.loanId != 0) {
433       (address walletBuyer, address protocolOwnerBuyer) =
  ↳ GenericLogic.getMainWallet(
434         _walletRegistry,
435         buyer
436       );
437
438       // Change the address of the buyer to the UnlockdWallet
439       buyer = walletBuyer;
440       // Block the asset
441       IProtocolOwner(protocolOwnerBuyer).setLoanId(signAuction.
  ↳ assetId, loan.loanId);
442       // Activate the loan from the bidder
443       _loans[order.bid.loanId].activate();
444     }
```

Proof of Concept:

1. Finalize a liquidation auction.
2. Verify that the loan ID is not updated properly.

The following image displays the asset's state change during the above auction finalization process:



```
Asset owner: 0xD554847189167Cbb4fFe9430040bD10E0E395516
Asset loan ID: 0x613435c0076b5b5f77423d72c25a27423a1d25b7d7ca24850cfaee3bddfe3270
--------------------------
*** auction finalize success ***
--------------------------
Asset owner: 0xf7f2F5c97eA304a20B34b3aee4771ED7255e7cbD
Asset loan ID: 0x613435c0076b5b5f77423d72c25a27423a1d25b7d7ca24850cfaee3bddfe3270
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

The loan ID of the asset should be correctly updated in the finalize function to reflect the ownership and state changes.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 1e76e61.

4.6 (HAL-06) LACK OF ORACLE DATA FEED VALIDATION - MEDIUM (5.0)

Description:

It was identified that the AggregatorV3Interfacedata source is not sufficiently validated, as there is no check for stale price in the getAssetPrice() function of the ReserveOracle contract. The lack of validation might result in receiving outdated data in the time of big price movements.

Code Location:

The price data returned from the aggregator is not validated:

Listing 15: src/libraries/oracles/ReserveOracle.sol (Line 95)

```
85   function getAssetPrice(address priceFeedKey) external view
 ↳ override returns (uint256) {
86     if (priceFeedKey == address(0)) revert Errors.
 ↳ InvalidPriceFeedKey();
87     AggregatorV3Interface aggregator = _priceFeedMap[priceFeedKey
 ↳ ];
88
89     if (priceFeedKey == BASE_CURRENCY) {
90       return BASE_CURRENCY_UNIT;
91     }
92
93     if (address(aggregator) == address(0)) revert Errors.
 ↳ InvalidAggregator();
94
95     (, int256 _price, , , ) = aggregator.latestRoundData();
96
97     return uint256(_price);
98   }
```

The aggregator returns additional data that can be used for price validation:

Listing 16: AggregatorV3Interface.sol

```
function latestRoundData()
  external
  view
  returns (
    uint80 roundId,
    int256 answer,
    uint256 startedAt,
    uint256 updatedAt,
    uint80 answeredInRound
  );
}
```

Proof of Concept:

1. The user tries to get a loan without enough collateral.
2. During the future loan state validation, the reserve oracle returns a stale price.
3. This results in invalid health factor data because it is calculated based on the outdated reserve unit price.
4. Because of the inaccurate health factor, the future loan state health check is passed when they should not.

BVSS:

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to validate the updatedAt parameter of the received price data to avoid stale prices.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 764c362.

4.7 (HAL-07) IMPROPER RESERVOIR WITHDRAWERC20 IMPLEMENTATION - MEDIUM (5.0)

Description:

It was identified that the withdrawERC20 function in the ReservoirAdapter has improper implementation as it withdraws the Ether from the contract. Therefore, it is not possible to withdraw the ERC20 tokens from the contract.

Note that the ReservoirAdapter contract is not upgradeable. And therefore, the received tokens will be stuck in the contracts indefinitely.

Also note that, only the Protocol can use these function, and not the Protocol Admin.

Code Location:

The withdrawERC20 uses the same implementation as the withdraw function:

Listing 17: src/protocol/adapters/ReservoirAdapter.sol (Lines 180-186)

```
  function withdraw(address payable _to) external onlyProtocol {
    (bool sent, ) = _to.call{value: address(this).balance}('');
    if (sent == false) revert Errors.UnsuccessfulExecution();
  }

  function withdrawERC20(address payable _to) external onlyProtocol {
    (bool sent, ) = _to.call{value: address(this).balance}('');
    if (sent == false) revert Errors.UnsuccessfulExecution();
  }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to fix the withdrawERC20 function to enable the protocol to withdraw the ERC20 tokens from the ReservoirAdapter contract.

It is also recommended to review if the authorization of the functions is aligned with their business requirements.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 3f6e82a.

4.8 (HAL-08) RESERVE STATE IS NOT UPDATED IN CREATE - LOW (2.5)

Description:

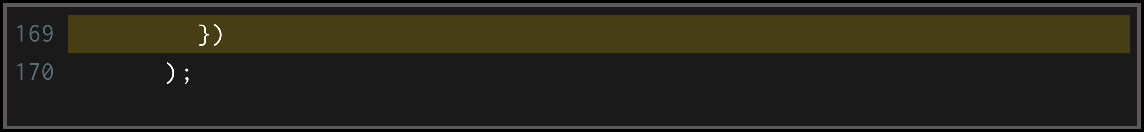
It was identified that the reserve state is not updated in the create() function of the Market module. If the auction is created with an NFT that is in a loan, a health check is executed to check the loan's state. Without updating to the latest borrow index, the data might be outdated, and the health check might be passed even without enough collateral.

Code Location:

If the auction is created with an NFT that is in a loan, the reserve state should be updated to reflect the latest state during the loan state validation:

Listing 18: src/protocol/modules/Market.sol (Lines 161-169)

```
148     } else {
149       if (loan.uToken != uToken) {
150         revert Errors.InvalidUToken();
151       }
152
153       if (loan.owner != msgSender) {
154         revert Errors.InvalidLoanOwner();
155       }
156
157       if (signMarket.loan.totalAssets == loan.totalAssets) {
158         revert Errors.LoanNotUpdated();
159       }
160
161       ValidationLogic.validateFutureLoanState(
162         ValidationLogic.ValidateLoanStateParams({
163           user: msgSender,
164           amount: config.startAmount,
165           price: signMarket.assetPrice,
166           reserveOracle: _reserveOracle,
167           reserve: IUToken(loan.uToken).getReserve(),
168           loanConfig: signMarket.loan
```



```
            })
        );
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended that the reserve state be updated to reflect the latest borrow index during the health check in the `create()` function.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 3aaad20.

4.9 (HAL-09) DANGEROUS REENTRANT PATTERNS - LOW (2.5)

Description:

It’s been identified several operations that involve the transference of ERC721 tokens, these tokens when are transferred to other accounts, in case these accounts contain smart contracts on them, it is necessary to implement specific functions such as onERC1155Received and onERC1155BatchReceived in order to accept the received tokens without reverting the transactions.

This kind of execution pattern generates reentrancy opportunities that can be exploited in case of critical parts of the storage are not updated properly before the reentrant code is executed, which could create inconsistencies between an amount of assets associated to a smart contract and a value stored in its internal storage, for instance.

Code Location:

The Checks-Effects-Interactions pattern is violated in the finalize() function:

Listing 19: src/protocol/modules/Auction.sol (Lines 462-467)

```
    IProtocolOwner(protocolOwner).changeOwner(
      signAuction.collection,
      signAuction.tokenId,
      // We send the asset to
      buyer
    );

    // The start amount it was payed as a debt
    uint256 amount = order.bid.amountOfDebt + order.bid.amountToPay - order.offer.startAmount;
    loan.underlyingAsset.safeTransfer(order.owner, amount);
    // Remove the order
    delete _orders[orderId];

```

```
475     // Check the messe it's correct
476     if (_loans[loan.loanId].totalAssets != signAuction.loan.totalAssets + 1) {
477       revert Errors.TokenAssetsMismatch();
478     }
479     if (signAuction.loan.totalAssets == 0) {
480       // If there is only one we can remove the loan
481       delete _loans[offerLoanId];
482     } else {
483       // Activate loan
484       loan.activate();
485       loan.totalAssets = signAuction.loan.totalAssets;
486     }
```

The Checks-Effects-Interactions pattern is violated in the claim() function:

Listing 20: src/protocol/modules/Market.sol (Lines 517-521)

```
517       IProtocolOwner(delegationOwnerOwner).changeOwner(
518         signMarket.collection,
519         signMarket.tokenId,
520         buyer
521       );
522
523       delete _orders[order.orderId];
```

The Checks-Effects-Interactions pattern is violated in the buyNow() function:

Listing 21: src/protocol/modules/Market.sol (Line 770)

```
770     IProtocolOwner(protocolOwner).changeOwner(signMarket.collection, signMarket.tokenId, buyer);
771
772     // We remove the current order asociated to this asset
773     delete _orders[orderId];
774
775     if (_loans[loan.loanId].totalAssets != signMarket.loan.totalAssets + 1) {
776       revert Errors.TokenAssetsMismatch();
777     }
```

```
    // We check the status
    if (signMarket.loan.totalAssets == 0) {
      // Remove the loan because doesn't have more assets
      delete _loans[loan.loanId];
    } else {
      // We update the counter
      _loans[loan.loanId].totalAssets = signMarket.loan.totalAssets;
      _loans[loan.loanId].activate();
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to always follow the Checks-Effects-Interactions pattern, which involves performing all necessary checks and updating internal state before interacting with any external contracts.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 118ab97.

4.10 (HAL-10) IMPROPER WITHDRAW LIQUIDITY CHECK - LOW (2.5)

Description:

It was identified that the withdraw() function in the UToken contract employed an improper available liquidity check. If the type(uint256).max value is used as the amount parameter, the function interprets it to withdraw the user's entire balance. However, it was identified that, in this case, the function tries to compare the available liquidity to the parameter value and not to the user's balance.

Code Location:

Listing 22: src/protocol/UToken.sol (Lines 180-186)

```
 function withdraw(uint256 amount, address to) external nonReentrant isActive returns (uint256) {
   Errors.verifyNotZero(to);
   Errors.verifyNotZero(amount);
   uint256 userBalance = this.balanceOf(_msgSender());
   if (amount > userBalance) {
     revert Errors.AmountExceedsBalance();
   }
   uint256 amountToWithdraw = amount;

   if (amount == type(uint256).max) {
     amountToWithdraw = userBalance;
   }
   uint256 availableLiquidity = super.totalSupply();
   if (amount > availableLiquidity) {
     revert Errors.NotEnoughLiquidity();
   }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to alter the function to compare the available liquidity to the amountToWithdraw variable.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 3f6e82a by reworking the UToken contract.

4.11 (HAL-11) MISSING ZERO ADDRESS CHECKS - INFORMATIONAL (0.8)

Description:

It was identified that the several parameters in the contracts lack zero address validation.

Code Location:

src/protocol/DebtToken.sol
- Line 37: setUToken is missing zero address checks for uToken.

src/protocol/UToken.sol
- Line 37: initialize is missing zero address checks for treasury.

src/deployer/DeployUTokenConfig.sol
- Line 28: constructor is missing zero address checks for admin, adminUpdater and aclManager.

src/deployer/DeployUToken.sol
- Line 27: constructor is missing zero address checks for admin, aclManager.

src/deployer/DeployPeriphery.sol
- Line 28: constructor is missing zero address checks for adminUpdater, aclManager.

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

It is recommended to add zero address validation for the listed parameters.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit 3f6e82a. The UToken, DebtToken contracts were reworked.

4.12 (HAL-12) MISSING EVENTS FOR CONTRACT OPERATIONS - INFORMATIONAL (0.8)

Description:

It was identified that several admin functions from the ReserveOracle and Manager contracts do not emit any events. As a result, blockchain monitoring systems might not be able to timely detect suspicious behaviors related to these functions.

src/libraries/oracles/ReserveOracle.sol
- Line 67: removeAggregator

src/protocol/modules/Manager.sol
- Line 118: emergencyFreezeLoan
- Line 127: emergencyActiveLoan
- Line 137: emergencyUpdateEndTimeAuction

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Consider adding events for all important operations to help monitor the contracts and detect suspicious behavior. A monitoring system that tracks relevant events would allow the timely detection of compromised system components.

Remediation Plan:

SOLVED: The Unlockd Finance team solved the issue in commit dcff6b7 by reviewing the management functions and adding events where it deemed advantageous.

AUTOMATED TESTING

5.1 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.
The security team assessed all findings identified by the Slither software, however, findings with severity Information and Optimization are not included in the below results for the sake of report readability.

Results:

src/utils/BlockContext.sol
Slither did not identify any vulnerabilities in the contract.

src/protocol/DebtToken.sol

Slither results for DebtToken.sol	
Finding	**Impact**
DebtToken.setUToken(address).uToken (src/protocol/DebtToken.sol#54) lacks a zero-check on : - _uToken = uToken (src/protocol/DebtToken.sol#55)	Low
End of table for DebtToken.sol	

src/protocol/UToken.sol

Slither results for UToken.sol	
Finding	**Impact**
UToken.repayOnBelhalf(bytes32,uint256,address,address) (src/protocol/UToken.sol#186-216) uses arbitrary from in transferFrom: IERC20(_reserve.underlyingAsset).safeTransferFrom(from,address(this),amount) (src/protocol/UToken.sol#196)	High
UToken._decimals (src/protocol/UToken.sol#31) is never initialized. It is used in: - UToken.decimals() (src/protocol/UToken.sol#329-331)	High
UToken.updateStateReserve() (src/protocol/UToken.sol#308-310) ignores return value by _reserve.updateState() (src/protocol/UToken.sol#309)	Medium
UToken.borrowOnBelhalf(bytes32,uint256,address,address) (src/protocol/UToken.sol#148-184) ignores return value by IDebtToken(_reserve.debtTokenAddress).mint(loanId,onBehalfOf,amount,_reserve.variableBorrowIndex) (src/protocol/UToken.sol#165-170)	Medium
UToken.setTreasuryAddress(address).treasury (src/protocol/UToken.sol#324) lacks a zero-check on : - _treasury = treasury (src/protocol/UToken.sol#326)	Low
UToken.initialize(address,address,address,address,address,uint8,uint16,string,string).treasury (src/protocol/UToken.sol#48) lacks a zero-check on : - _treasury = treasury (src/protocol/UToken.sol#58)	Low
UToken.setTreasury(address).treasury (src/protocol/UToken.sol#223) lacks a zero-check on : - _treasury = treasury (src/protocol/UToken.sol#225)	Low
End of table for UToken.sol	

AUTOMATED TESTING

src/protocol/Unlockd.sol

Slither results for Unlockd.sol	
Finding	**Impact**
Unlockd.constructor(address,address).aclManager (src/protocol/Unlockd.sol#11) lacks a zero-check on : - _aclManager = aclManager (src/protocol/Unlockd.sol#17)	Low
End of table for Unlockd.sol	

src/protocol/adapters/ReservoirAdapter.sol

Slither results for ReservoirAdapter.sol	
Finding	**Impact**
ReservoirAdapter.sell(IMarketAdapter.SellParams) (src/protocol/adapters/ReservoirAdapter.sol#54-67) uses arbitrary from in transferFrom: IERC20(params.underlyingAsset).safeTransferFrom(params.wallet,msg.sender,params.marketPrice) (src/protocol/adapters/ReservoirAdapter.sol#66)	High
ReservoirAdapter._rawExec(address,uint256,bytes) (src/protocol/adapters/ReservoirAdapter.sol#99-104) sends eth to arbitrary user Dangerous calls: - (sent) = address(to).call{value: value}(data) (src/protocol/adapters/ReservoirAdapter.sol#102)	High
ReservoirAdapter.withdrawERC20(address) (src/protocol/adapters/ReservoirAdapter.sol#94-97) sends eth to arbitrary user Dangerous calls: - (sent) = _to.call{value: address(this).balance}() (src/protocol/adapters/ReservoirAdapter.sol#95)	High
ReservoirAdapter.withdraw(address) (src/protocol/adapters/ReservoirAdapter.sol#89-92) sends eth to arbitrary user Dangerous calls: - (sent) = _to.call{value: address(this).balance}() (src/protocol/adapters/ReservoirAdapter.sol#90)	High
ReservoirAdapter.withdrawERC20(address) (src/protocol/adapters/ReservoirAdapter.sol#94-97) uses a dangerous strict equality: - sent == false (src/protocol/adapters/ReservoirAdapter.sol#96)	Medium

Finding	Impact
ReservoirAdapter.withdraw(address) (src/protocol/adapters/ReservoirAdapter.sol#89-92) uses a dangerous strict equality: - sent == false (src/protocol/adapters/ReservoirAdapter.sol#91)	Medium
ReservoirAdapter.buy(IMarketAdapter.BuyParams) (src/protocol/adapters/ReservoirAdapter.sol#73-87) ignores return value by IERC20(params.underlyingAsset).approve(params.marketApproval,params.marketPrice) (src/protocol/adapters/ReservoirAdapter.sol#77)	Medium
ReservoirAdapter.sell(IMarketAdapter.SellParams) (src/protocol/adapters/ReservoirAdapter.sol#54-67) ignores return value by IProtocolOwner(params.protocolOwner).execTransaction(params.to,params.value,params.data,0,0,0,address(0),address(0)) (src/protocol/adapters/ReservoirAdapter.sol#55-64)	Medium
ReservoirAdapter.withdraw(address)._to (src/protocol/adapters/ReservoirAdapter.sol#89) lacks a zero-check on : - (sent) = _to.call{value: address(this).balance}() (src/protocol/adapters/ReservoirAdapter.sol#90)	Low
ReservoirAdapter.constructor(address,address,address).reservoir (src/protocol/adapters/ReservoirAdapter.sol#37) lacks a zero-check on : - RESERVOIR = reservoir (src/protocol/adapters/ReservoirAdapter.sol#38)	Low
ReservoirAdapter.constructor(address,address,address).aclManager (src/protocol/adapters/ReservoirAdapter.sol#37) lacks a zero-check on : - _aclManager = aclManager (src/protocol/adapters/ReservoirAdapter.sol#40)	Low
ReservoirAdapter.constructor(address,address,address).eth (src/protocol/adapters/ReservoirAdapter.sol#37) lacks a zero-check on : - ETH_RESERVOIR = eth (src/protocol/adapters/ReservoirAdapter.sol#39)	Low

Finding	Impact
ReservoirAdapter.withdrawERC20(address)._to (src/protocol/adapters/ReservoirAdapter.sol#94) lacks a zero-check on : - (sent) = _to.call{value: address(this).balance}() (src/protocol/adapters/ReservoirAdapter.sol#95)	Low
End of table for ReservoirAdapter.sol	

src/protocol/modules/Market.sol

Slither results for Market.sol	
Finding	**Impact**
Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) uses arbitrary from in transferFrom: IERC20(loan.underlyingAsset).safeTransferFrom(msgSender,address(this),amountToPay) (src/protocol/modules/Market.sol#324)	High
Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) uses arbitrary from in transferFrom: IERC20(underlyingAsset).safeTransferFrom(msgSender,address(this),amountToPay) (src/protocol/modules/Market.sol#670)	High

Finding	Impact
Reentrancy in Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#650) - IERC20(underlyingAsset).safeTransferFrom(msgSender,address(this),amountToPay) (src/protocol/modules/Market.sol#670) - OrderLogic.borrowByBidder(OrderLogic.BorrowByBidderParams(newLoanId,msgSender,uToken,amountOfDebt,signMarket.assetPrice,signMarket.assetLtv)) (src/protocol/modules/Market.sol#693-702) - IProtocolOwner(delegationOwnerBuyer).setLoanId(order.offer.assetId,newLoanId) (src/protocol/modules/Market.sol#715) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,uToken,underlyingAsset,order.bid.amountOfDebt,order.bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#722-733) - totalAmount = OrderLogic.repayDebtToSell(order,OrderLogic.RepayDebtToSellParams(_reserveOracle,underlyingAsset,uToken,totalAmount,signMarket.loan.aggLoanPrice,signMarket.loan.aggLtv),reserve) (src/protocol/modules/Market.sol#742-753) - IERC20(underlyingAsset).safeTransfer(order.owner,totalAmount) (src/protocol/modules/Market.sol#757) - IProtocolOwner(delegationOwner_scope_0).changeOwner(signMarket.collection,signMarket.tokenId,buyer) (src/protocol/modules/Market.sol#763) State variables written after the call(s): - delete _loans[loan.loanId] (src/protocol/modules/Market.sol#774) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) - _loans[loan.loanId].totalAssets = signMarket.loan.totalAssets (src/protocol/modules/Market.sol#777) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross	Medium

Finding	Impact
Reentrancy in Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266): External calls: - IUToken(loanUToken).updateStateReserve() (src/protocol/modules/Market.sol#243) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(bid.loanId,bid.buyer,_reserveOracle,loanUToken,loan.underlyingAsset,bid.amountOfDebt,bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#245-256) State variables written after the call(s): - delete _loans[bid.loanId] (src/protocol/modules/Market.sol#259) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) - delete _orders[orderId] (src/protocol/modules/Market.sol#263) CoreStorage._orders (src/libraries/storage/CoreStorage.sol#57) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) - Market.getBuyNowPrice(bytes32,address,uint256,uint256) (src/protocol/modules/Market.sol#80-96) - Market.getMinBidPrice(bytes32,address,uint256,uint256) (src/protocol/modules/Market.sol#58-71) - Market.getOrder(bytes32) (src/protocol/modules/Market.sol#47-49)	Medium

Finding	Impact
Reentrancy in Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#650) - IERC20(underlyingAsset).safeTransferFrom(msgSender,address(this),amountToPay) (src/protocol/modules/Market.sol#670) - OrderLogic.borrowByBidder(OrderLogic.BorrowByBidderParams(newLoanId,msgSender,uToken,amountOfDebt,signMarket.assetPrice,signMarket.assetLtv)) (src/protocol/modules/Market.sol#693-702) - IProtocolOwner(delegationOwnerBuyer).setLoanId(order.offer.assetId,newLoanId) (src/protocol/modules/Market.sol#715) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,uToken,underlyingAsset,order.bid.amountOfDebt,order.bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#722-733) State variables written after the call(s): - delete _loans[order.bid.loanId] (src/protocol/modules/Market.sol#737) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210)	Medium

Finding	Impact
Reentrancy in Market.cancelClaim(bool,bytes32,DataTypes.SignMarket, DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#557) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,uToken,loan.underlyingAsset,order.bid.amountOfDebt,order.bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#579-590) State variables written after the call(s): - delete _loans[order.bid.loanId] (src/protocol/modules/Market.sol#594) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) - delete _orders[order.orderId] (src/protocol/modules/Market.sol#599) CoreStorage._orders (src/libraries/storage/CoreStorage.sol#57) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) - Market.getBuyNowPrice(bytes32,address,uint256,uint256) (src/protocol/modules/Market.sol#80-96)	Medium

Finding	Impact
Reentrancy in Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#306) - IERC20(loan.underlyingAsset).safeTransferFrom(msgSender,address(this),amountToPay) (src/protocol/modules/Market.sol#324) - OrderLogic.borrowByBidder(OrderLogic.BorrowByBidderParams(loanId,msgSender,uToken,amountOfDebt,signMarket.assetPrice,signMarket.assetLtv)) (src/protocol/modules/Market.sol#335-344) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,uToken,loan.underlyingAsset,order.bid.amountOfDebt,order.bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#362-373) State variables written after the call(s): - delete _loans[order.bid.loanId] (src/protocol/modules/Market.sol#377) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) - order.countBids ++ (src/protocol/modules/Market.sol#381) CoreStorage._orders (src/libraries/storage/CoreStorage.sol#57) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrd	Medium

Finding	Impact
Reentrancy in Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#433) - totalAmount = OrderLogic.repayDebtToSell(order,OrderLogic.RepayDebtToSellParams(_reserveOracle,underlyingAsset,uToken,totalAmount,signMarket.loan.aggLoanPrice,signMarket.loan.aggLtv),reserve) (src/protocol/modules/Market.sol#453-464) - IERC20(underlyingAsset).safeTransfer(order.owner,totalAmount) (src/protocol/modules/Market.sol#467) - IProtocolOwner(buyerDelegationOwner).setLoanId(order.offer.assetId,order.bid.loanId) (src/protocol/modules/Market.sol#486) - IProtocolOwner(delegationOwnerOwner).changeOwner(signMarket.collection,signMarket.tokenId,buyer) (src/protocol/modules/Market.sol#514-518) State variables written after the call(s): - delete _orders[order.orderId] (src/protocol/modules/Market.sol#520) CoreStorage._orders (src/libraries/storage/CoreStorage.sol#57) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) - Market.getBuyNowPrice(bytes32,address,uint256,uint256) (src/protocol/modules/Market.sol#80-96) - Market.getMinBidPrice(bytes32,address,uint256,uint256) (src/protocol/modules/Market.sol#58-71) - Market.getOrder(bytes32) (src/protocol/modules/Market.sol#47-49)	Medium

Finding	Impact
Reentrancy in Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#433) - totalAmount = OrderLogic.repayDebtToSell(order,OrderLogic.RepayDebtToSellParams(_reserveOracle,underlyingAsset,uToken,totalAmount,signMarket.loan.aggLoanPrice,signMarket.loan.aggLtv),reserve) (src/protocol/modules/Market.sol#453-464) - IERC20(underlyingAsset).safeTransfer(order.owner,totalAmount) (src/protocol/modules/Market.sol#467) - IProtocolOwner(buyerDelegationOwner).setLoanId(order.offer.assetId,order.bid.loanId) (src/protocol/modules/Market.sol#486) State variables written after the call(s): - delete _loans[loanId] (src/protocol/modules/Market.sol#500) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) - _loans[loanId].totalAssets = signMarket.loan.totalAssets (src/protocol/modules/Market.sol#503) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) - Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789) - Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266) - Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608) - Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) - Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature)	Medium

Finding	Impact
Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature).delegationOwner_scope_0 (src/protocol/modules/Market.sol#759) is a local variable never initialized	Medium
Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature).buyerDelegationOwner (src/protocol/modules/Market.sol#471) is a local variable never initialized	Medium
Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature).delegationOwnerBuyer (src/protocol/modules/Market.sol#674) is a local variable never initialized	Medium
Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524) ignores return value by ValidationLogic.validateFutureLoanState(ValidationLogic.ValidateLoanStateParams(order.owner,totalAmount,signMarket.assetPrice,_reserveOracle,IUToken(loan.uToken).getReserve(),signMarket.loan)) (src/protocol/modules/Market.sol#441-450)	Medium
Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210) ignores return value by ValidationLogic.validateFutureLoanState(ValidationLogic.ValidateLoanStateParams(msgSender,config.startAmount,signMarket.assetPrice,_reserveOracle,IUToken(loan.uToken).getReserve(),signMarket.loan)) (src/protocol/modules/Market.sol#161-170)	Medium
Market.cancel(bytes32).bid (src/protocol/modules/Market.sol#225) shadows: - Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393) (function)	Low

Finding	Impact
Reentrancy in Market.cancelClaim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#526-608): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#557) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,uToken,loan.underlyingAsset,order.bid.amountOfDebt,order.bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#579-590) Event emitted after the call(s): - MarketCancelBid(order.offer.loanId,order.orderId,signMarket.assetId,totalAmount,order.owner) (src/protocol/modules/Market.sol#601-607)	Low
Reentrancy in Market.bid(bytes32,uint128,uint128,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#276-393): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#306) - IERC20(loan.underlyingAsset).safeTransferFrom(msgSender,address(this),amountToPay) (src/protocol/modules/Market.sol#324) - OrderLogic.borrowByBidder(OrderLogic.BorrowByBidderParams(loanId,msgSender,uToken,amountOfDebt,signMarket.assetPrice,signMarket.assetLtv)) (src/protocol/modules/Market.sol#335-344) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,uToken,loan.underlyingAsset,order.bid.amountOfDebt,order.bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#362-373) Event emitted after the call(s): - MarketBid(loanId,order.orderId,order.offer.assetId,totalAmount,msgSender) (src/protocol/modules/Market.sol#392)	Low
Reentrancy in Market.create(address,DataTypes.OrderType,IMarketModule.CreateOrderInput,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#107-210): External calls: - IProtocolOwner(delegationOwner).setLoanId(signMarket.assetId,loan.loanId) (src/protocol/modules/Market.sol#147) Event emitted after the call(s): - MarketCreated(signMarket.loan.loanId,orderId,signMarket.assetId,signMarket.collection,signMarket.tokenId) (src/protocol/modules/Market.sol#203-209)	Low

Finding	Impact
Reentrancy in Market.claim(bool,bytes32,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#402-524): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#433) - totalAmount = OrderLogic.repayDebtToSell(order,OrderLogic.RepayDebtToSellParams(_reserveOracle,underlyingAsset,uToken,totalAmount,signMarket.loan.aggLoanPrice,signMarket.loan.aggLtv),reserve) (src/protocol/modules/Market.sol#453-464) - IERC20(underlyingAsset).safeTransfer(order.owner,totalAmount) (src/protocol/modules/Market.sol#467) - IProtocolOwner(buyerDelegationOwner).setLoanId(order.offer.assetId,order.bid.loanId) (src/protocol/modules/Market.sol#486) - IProtocolOwner(delegationOwnerOwner).changeOwner(signMarket.collection,signMarket.tokenId,buyer) (src/protocol/modules/Market.sol#514-518) Event emitted after the call(s): - MarketClaim(loanId,order.orderId,signMarket.assetId,totalAmount,msgSender) (src/protocol/modules/Market.sol#522)	Low

Finding	Impact
Reentrancy in Market.buyNow(bool,bytes32,uint256,uint256,DataTypes.SignMarket,DataTypes.EIP712Signature) (src/protocol/modules/Market.sol#619-789): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/Market.sol#650) - IERC20(underlyingAsset).safeTransferFrom(msgSender,address(this),amountToPay) (src/protocol/modules/Market.sol#670) - OrderLogic.borrowByBidder(OrderLogic.BorrowByBidderParams(newLoanId,msgSender,uToken,amountOfDebt,signMarket.assetPrice,signMarket.assetLtv)) (src/protocol/modules/Market.sol#693-702) - IProtocolOwner(delegationOwnerBuyer).setLoanId(order.offer.assetId,newLoanId) (src/protocol/modules/Market.sol#715) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,uToken,underlyingAsset,order.bid.amountOfDebt,order.bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#722-733) - totalAmount = OrderLogic.repayDebtToSell(order,OrderLogic.RepayDebtToSellParams(_reserveOracle,underlyingAsset,uToken,totalAmount,signMarket.loan.aggLoanPrice,signMarket.loan.aggLtv),reserve) (src/protocol/modules/Market.sol#742-753) - IERC20(underlyingAsset).safeTransfer(order.owner,totalAmount) (src/protocol/modules/Market.sol#757) - IProtocolOwner(delegationOwner_scope_0).changeOwner(signMarket.collection,signMarket.tokenId,buyer) (src/protocol/modules/Market.sol#763) Event emitted after the call(s): - MarketBuyNow(signMarket.loan.loanId,orderId,signMarket.assetId,totalAmount,msgSender) (src/protocol/modules/Market.sol#781-787)	Low
Reentrancy in Market.cancel(bytes32) (src/protocol/modules/Market.sol#216-266): External calls: - IUToken(loanUToken).updateStateReserve() (src/protocol/modules/Market.sol#243) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(bid.loanId,bid.buyer,_reserveOracle,loanUToken,loan.underlyingAsset,bid.amountOfDebt,bid.amountToPay,reserve)) (src/protocol/modules/Market.sol#245-256) Event emitted after the call(s): - MarketCancelAuction(loan.loanId,orderId,order.owner) (src/protocol/modules/Market.sol#265)	Low

Finding	Impact
End of table for Market.sol	

src/protocol/modules/BuyNow.sol

Slither results for BuyNow.sol	
Finding	**Impact**
BuyNow.buy(address,address,uint256,DataTypes.SignBuyNow,DataTypes.EIP712Signature) (src/protocol/modules/BuyNow.sol#68-136) uses arbitrary from in transferFrom: reserve.underlyingAsset.safeTransferFrom(msgSender,marketAdapter,amount) (src/protocol/modules/BuyNow.sol#91)	High
BuyNow.buy(address,address,uint256,DataTypes.SignBuyNow,DataTypes.EIP712Signature).vars (src/protocol/modules/BuyNow.sol#80) is a local variable never initialized	Medium
Reentrancy in BuyNow.buy(address,address,uint256,DataTypes.SignBuyNow,DataTypes.EIP712Signature) (src/protocol/modules/BuyNow.sol#68-136): External calls: - IUToken(uToken).updateStateReserve() (src/protocol/modules/BuyNow.sol#87) - vars.loanId = _borrowLoan(msgSender,marketAdapter,amount,uToken,protocolOwner,reserve.underlyingAsset,signBuyMarket) (src/protocol/modules/BuyNow.sol#95-103) - IUToken(uToken).borrowOnBelhalf(loanId,amountNeeded,marketAdapter,msgSender) (src/protocol/modules/BuyNow.sol#167) - IProtocolOwner(protocolOwner).setLoanId(signBuyMarket.asset.assetId,loanId) (src/protocol/modules/BuyNow.sol#169) - vars.realCost = IMarketAdapter(marketAdapter).buy(IMarketAdapter.BuyParams(wallet,signBuyMarket.underlyingAsset,signBuyMarket.marketPrice,signBuyMarket.marketApproval,signBuyMarket.to,signBuyMarket.value,signBuyMarket.data)) (src/protocol/modules/BuyNow.sol#110-120) Event emitted after the call(s): - BuyNowPayLater(vars.loanId,signBuyMarket.asset.collection,signBuyMarket.asset.tokenId,vars.realCost,amount) (src/protocol/modules/BuyNow.sol#129-135)	Low
End of table for BuyNow.sol	

src/protocol/modules/Installer.sol

Slither results for Installer.sol	
Finding	**Impact**
Installer.installModules(address[]) (src/protocol/modules/Installer.sol#18-42) has external calls inside a loop: newModuleId = BaseCoreModule(moduleAddr).moduleId() (src/protocol/modules/Installer.sol#22)	Low
Installer.installModules(address[]) (src/protocol/modules/Installer.sol#18-42) has external calls inside a loop: moduleVersion = BaseCoreModule(moduleAddr).moduleVersion() (src/protocol/modules/Installer.sol#23)	Low
End of table for Installer.sol	

src/protocol/modules/Auction.sol

Slither results for Auction.sol	
Finding	**Impact**
Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398) uses arbitrary from in transferFrom: underlyingAsset.safeTransferFrom(msgSender,address(this),amount) (src/protocol/modules/Auction.sol#368)	High
Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) uses arbitrary from in transferFrom: loan.underlyingAsset.safeTransferFrom(msgSender,address(this),amountToPay) (src/protocol/modules/Auction.sol#219)	High

Finding	Impact
Reentrancy in Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485): External calls: - IProtocolOwner(protocolOwnerBuyer).setLoanId(signAuction.assetId,loan.loanId) (src/protocol/modules/Auction.sol#441) - IProtocolOwner(protocolOwner).changeOwner(signAuction.collection,signAuction.tokenId,buyer) (src/protocol/modules/Auction.sol#450-455) State variables written after the call(s): - loan.totalAssets = signAuction.loan.totalAssets (src/protocol/modules/Auction.sol#470) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) - Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485) - Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398) - delete _loans[offerLoanId] (src/protocol/modules/Auction.sol#473) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) - Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485) - Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398) - delete _orders[orderId] (src/protocol/modules/Auction.sol#461)CoreStorage._orders (src/libraries/storage/CoreStorage.sol#57) can be used in cross function reentrancies: - Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) - Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485) - Auction.getMinBidPriceAuction(bytes32,address,uint256,uint256) (src/protocol/modules/Auction.sol#90-103) - Auction.getOrderAuction(bytes32) (src/protocol/modules/Auction.sol#109-111) - Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398)	Medium

Finding	Impact
Reentrancy in Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317): External calls: - utoken.updateStateReserve() (src/protocol/modules/Auction.sol#134) - OrderLogic.borrowByBidder(OrderLogic.BorrowByBidderParams(loanId,msgSender,address(utoken),amountOfDebt,signAuction.assetPrice,signAuction.assetLtv)) (src/protocol/modules/Auction.sol#231-240) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,address(utoken),loan.underlyingAsset,order.bid.amountOfDebt,amountToPayBuyer,reserve)) (src/protocol/modules/Auction.sol#284-295) State variables written after the call(s): - delete _loans[loanId_] (src/protocol/modules/Auction.sol#301) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) - Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485) - Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398)	Medium

Finding	Impact
Reentrancy in Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398): External calls: - IUToken(utoken).updateStateReserve() (src/protocol/modules/Auction.sol#349) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,utoken,underlyingAsset,order.bid.amountOfDebt,order.offer.startAmount + bidderBonus,reserve)) (src/protocol/modules/Auction.sol#371-382) State variables written after the call(s): - delete _loans[order.bid.loanId] (src/protocol/modules/Auction.sol#386) CoreStorage._loans (src/libraries/storage/CoreStorage.sol#55) can be used in cross function reentrancies: - Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) - Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485) - Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398)	Medium

Finding	Impact
Reentrancy in Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317): External calls: - utoken.updateStateReserve() (src/protocol/modules/Auction.sol#134) - OrderLogic.borrowByBidder(OrderLogic.BorrowByBidderParams(loanId,msgSender,address(utoken),amountOfDebt,signAuction.assetPrice,signAuction.assetLtv)) (src/protocol/modules/Auction.sol#231-240) - OrderLogic.repayOwnerDebt(OrderLogic.RepayOwnerDebtParams(loan.loanId,order.owner,address(utoken),loan.underlyingAsset,minBid)) (src/protocol/modules/Auction.sol#261-269) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,address(utoken),loan.underlyingAsset,order.bid.amountOfDebt,amountToPayBuyer,reserve)) (src/protocol/modules/Auction.sol#284-295) State variables written after the call(s): - order.countBids ++ (src/protocol/modules/Auction.sol#306)CoreStorage._orders (src/libraries/storage/CoreStorage.sol#57) can be used in cross function reentrancies: - Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) - Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485) - Auction.getMinBidPriceAuction(bytes32,address,uint256,uint256) (src/protocol/modules/Auction.sol#90-103) - Auction.getOrderAuction(bytes32) (src/protocol/modules/Auction.sol#109-111) - Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398) - order.bid = DataTypes.Bid(loanId,amountToPay,amountOfDebt,msgSender) (src/protocol/modules/Auction.sol#309-314)CoreStorage._orders (src/libraries/storage/CoreStorage.sol#57) can be used in cross function reentrancies: - Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) - Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485) - Auction.getMinBidPriceAuction(bytes32,address,uint256,uint256) (src/protocol/modules/Auction.sol#90-103) - Auction.getOrderAuction(bytes32) (src/protocol/modules/Auction.sol#109-111) - Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398)	Medium

Finding	Impact
Reentrancy in Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398): External calls: - IUToken(utoken).updateStateReserve() (src/protocol/modules/Auction.sol#349) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,utoken,underlyingAsset,order.bid.amountOfDebt,order.offer.startAmount + bidderBonus,reserve)) (src/protocol/modules/Auction.sol#371-382) - IUToken(utoken).repayOnBelhalf(order.offer.loanId,minDebt,address(this),msgSender) (src/protocol/modules/Auction.sol#391) State variables written after the call(s): - delete _orders[orderId] (src/protocol/modules/Auction.sol#394)CoreStorage._orders (src/libraries/storage/CoreStorage.sol#57) can be used in cross function reentrancies: - Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) - Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485) - Auction.getMinBidPriceAuction(bytes32,address,uint256,uint256) (src/protocol/modules/Auction.sol#90-103) - Auction.getOrderAuction(bytes32) (src/protocol/modules/Auction.sol#109-111) - Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398)	Medium
Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature).loanId (src/protocol/modules/Auction.sol#220) is a local variable never initialized	Medium
Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) ignores return value by ValidationLogic.validateFutureHasUnhealtyLoanState(ValidationLogic.ValidateLoanStateParams(loan.owner,0,signAuction.assetPrice,_reserveOracle,reserve,signAuction.loan)) (src/protocol/modules/Auction.sol#161-170)	Medium

Finding	Impact
Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317) ignores return value by ValidationLogic.validateFutureHasUnhealtyLoanState(ValidationLogic.ValidateLoanStateParams(order.owner,order.bid.amountOfDebt + order.bid.amountToPay,signAuction.assetPrice,_reserveOracle,reserve,signAuction.loan)) (src/protocol/modules/Auction.sol#201-210)	Medium
Reentrancy in Auction.redeem(bytes32,uint256,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#326-398): External calls: - IUToken(utoken).updateStateReserve() (src/protocol/modules/Auction.sol#349) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,utoken,underlyingAsset,order.bid.amountOfDebt,order.offer.startAmount + bidderBonus,reserve)) (src/protocol/modules/Auction.sol#371-382) - IUToken(utoken).repayOnBelhalf(order.offer.loanId,minDebt,address(this),msgSender) (src/protocol/modules/Auction.sol#391) Event emitted after the call(s): - AuctionRedeem(order.offer.loanId,orderId,order.offer.assetId,totalAmount,msgSender) (src/protocol/modules/Auction.sol#397)	Low
Reentrancy in Auction.finalize(bytes32,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#406-485): External calls: - IProtocolOwner(protocolOwnerBuyer).setLoanId(signAuction.assetId,loan.loanId) (src/protocol/modules/Auction.sol#441) - IProtocolOwner(protocolOwner).changeOwner(signAuction.collection,signAuction.tokenId,buyer) (src/protocol/modules/Auction.sol#450-455) Event emitted after the call(s): - AuctionFinalize(offerLoanId,orderId,signAuction.assetId,order.offer.startAmount,amount,order.bid.buyer,order.owner) (src/protocol/modules/Auction.sol#476-484)	Low

Finding	Impact
Reentrancy in Auction.bid(uint128,uint128,DataTypes.SignAuction,DataTypes.EIP712Signature) (src/protocol/modules/Auction.sol#120-317): External calls: - utoken.updateStateReserve() (src/protocol/modules/Auction.sol#134) - OrderLogic.borrowByBidder(OrderLogic.BorrowByBidderParams(loanId,msgSender,address(utoken),amountOfDebt,signAuction.assetPrice,signAuction.assetLtv)) (src/protocol/modules/Auction.sol#231-240) - OrderLogic.repayOwnerDebt(OrderLogic.RepayOwnerDebtParams(loan.loanId,order.owner,address(utoken),loan.underlyingAsset,minBid)) (src/protocol/modules/Auction.sol#261-269) - OrderLogic.refundBidder(OrderLogic.RefundBidderParams(order.bid.loanId,order.bid.buyer,_reserveOracle,address(utoken),loan.underlyingAsset,order.bid.amountOfDebt,amountToPayBuyer,reserve)) (src/protocol/modules/Auction.sol#284-295) Event emitted after the call(s): - AuctionBid(loanId,order.orderId,order.offer.assetId,totalAmount,msgSender) (src/protocol/modules/Auction.sol#316)	Low
End of table for Auction.sol	

src/protocol/modules/SellNow.sol

Slither did not identify any vulnerabilities in the contract.

src/protocol/modules/Action.sol

Slither did not identify any vulnerabilities in the contract.

src/protocol/modules/Manager.sol

Slither did not identify any vulnerabilities in the contract.

src/deployer/DeployProtocol.sol

Slither did not identify any vulnerabilities in the contract.

src/deployer/DeployPeriphery.sol

Slither results for DeployPeriphery.sol	
Finding	**Impact**
DeployPeriphery.constructor(address,address).adminUpdater (src/deployer/DeployPeriphery.sol#28) lacks a zero-check on : - _adminUpdater = adminUpdater (src/deployer/DeployPeriphery.sol#29)	Low
DeployPeriphery.constructor(address,address).aclManager (src/deployer/DeployPeriphery.sol#28) lacks a zero-check on : - _aclManager = aclManager (src/deployer/DeployPeriphery.sol#30)	Low
End of table for DeployPeriphery.sol	

src/deployer/DeployUToken.sol

Slither results for DeployUToken.sol	
Finding	**Impact**
DeployUToken.constructor(address,address).admin (src/deployer/DeployUToken.sol#27) lacks a zero-check on : - _admin = admin (src/deployer/DeployUToken.sol#28)	Low
DeployUToken.constructor(address,address).aclManager (src/deployer/DeployUToken.sol#27) lacks a zero-check on : - _aclManager = aclManager (src/deployer/DeployUToken.sol#29)	Low
End of table for DeployUToken.sol	

src/deployer/DeployUTokenConfig.sol

Slither results for DeployUTokenConfig.sol	
Finding	**Impact**
DeployUTokenConfig.constructor(address,address,address).adminUpdater (src/deployer/DeployUTokenConfig.sol#28) lacks a zero-check on : - _adminUpdater = adminUpdater (src/deployer/DeployUTokenConfig.sol#30)	Low
DeployUTokenConfig.constructor(address,address,address).admin (src/deployer/DeployUTokenConfig.sol#28) lacks a zero-check on : - _admin = admin (src/deployer/DeployUTokenConfig.sol#29)	Low

Finding	Impact
DeployUTokenConfig.constructor(address,address,address).aclManager (src/deployer/DeployUTokenConfig.sol#28) lacks a zero-check on : - _aclManager = aclManager (src/deployer/DeployUTokenConfig.sol#31)	Low
End of table for DeployUTokenConfig.sol	

Results Summary:

The findings obtained as a result of the Slither scan were reviewed:
- The lack of zero-check on, should emit an event findings were added to the report.
- The uses timestamp for comparisons and has external calls inside loop informational findings were manually reviewed and determined false-positives.
- The uses arbitrary from in transferFrom, never initialized, reentrancy, sends eth to arbitrary user, uses a dangerous strict equality and ignores return value vulnerabilities were manually reviewed and determined false-positives.

THANK YOU FOR CHOOSING

// HALBORN