// HALBORN

Mori Finance - Mori Protocol

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **August 21st, 2023 - September 13th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	09/13/2023
0.2	Document Updates	09/12/2023
0.3	Draft Version	09/14/2023
0.4	Draft Review	09/15/2023
0.5	Draft Review	09/15/2023
1.0	Remediation Plan	09/25/2023
1.1	Remediation Plan Review	09/25/2023
1.2	Remediation Plan Review	09/25/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Mori is a next-generation native stable asset DeFi protocol built on Ethereum. It generates low-volatility stable assets without any loss by collateralizing ETH.

Mori Finance engaged Halborn to conduct a security assessment on their smart contracts beginning on August 21st, 2023 and ending on September 13th, 2023. The security assessment was scoped to the smart contracts provided in the mori-defi/mori-contracts GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided 3 weeks for the engagement and assigned a team of 1 full-time security engineer to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts 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, which were partially addressed by Mori Finance. The main one was the following:

- The missing cacheTwap() function was removed from the ITreasury interface.

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 walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs (MythX).
- Static Analysis of security for scoped contract, and imported functions (Slither).
- Testnet deployment (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) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 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. Project Name

- Repository: mori-defi/mori-contracts
- Commit ID: 2a6d4494f8930bfebd555553f432d03cc1bf6131
- Remediation Commit: b0054332a2d596706f134e7785df48c00cd5c987
- Smart contracts in scope:
    1. ChainlinkTwapOracleV3 (mori-contracts/contracts/ChainlinkTwapOracleV3.sol)
    2. Treasury (mori-contracts/contracts/Treasury.sol)
    3. StableCoinMath (mori-contracts/contracts/StableCoinMath.sol)
    4. MoriGateway (mori-contracts/contracts/MoriGateway.sol)
    5. Market (mori-contracts/contracts/Market.sol)
    6. LeveragedToken (mori-contracts/contracts/LeveragedToken.sol)
    7. FractionalToken (mori-contracts/contracts/FractionalToken.sol)

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) IMPLEMENTATION CONTRACT CAN BE INITIALIZED BY ANYONE	Low (2.5)	RISK ACCEPTED
(HAL-02) ACCESS CONTROL DOES NOT FOLLOW SECURITY BEST PRACTICES	Low (2.0)	RISK ACCEPTED
(HAL-03) MISSING FUNCTION IMPLEMENTATION	Informational (0.0)	SOLVED - 09/25/2023
(HAL-04) FLOATING PRAGMA	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) IMPLEMENTATION CONTRACT CAN BE INITIALIZED BY ANYONE - LOW (2.5)

The upgradeable contracts in scope contain the initializer function. However, the initializers are not being disabled in the contract's constructor.

This means that when an implementation is deployed in order to be used with a proxy, the implementation is left uninitialized, and therefore anyone can initialize it and take ownership of the contract. This is often used in phishing attacks.

BVSS:

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

Recommendation:

Call the _disableInitializers() function in the contract's constructor.

Remediation Plan:

RISK ACCEPTED: The Mori Finance team accepted the risk of this issue.

4.2 (HAL-02) ACCESS CONTROL DOES NOT FOLLOW SECURITY BEST PRACTICES - LOW (2.0)

Description:

The Market contract inherits from the OpenZeppelin's AccessControl library. However, this library does not follow some security best practices, for example, the DEFAULT_ADMIN_ROLE is also its own admin, meaning it has permissions to grant and revoke this role.

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (2.0)

Recommendation:

Consider following security best practices and OpenZeppelin's recommendations, and use the AccessControlDefaultAdminRules extension to enforce additional security measures over this role.

Remediation Plan:

RISK ACCEPTED: The Mori Finance team accepted the risk of this issue

4.3 (HAL-03) MISSING FUNCTION IMPLEMENTATION - INFORMATIONAL (0.0)

Description:

The interface for the Treasury contract contains the cacheTwap() function, which is not implemented in the contract itself. This will prevent the contract from properly compiling, unless the contract is marked as abstract.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Remove the function from the interface or implement the function in the contract.

Remediation Plan:

SOLVED: The Mori Finance team solved the issue in commit b005433 by removing the cacheTwap() function from the ITreasury interface.

4.4 (HAL-04) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Set the pragma to a fixed version.

Remediation Plan:

ACKNOWLEDGED: The Mori Finance team acknowledged the risk of this issue.

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the program for.

5.1 ACCESS CONTROL

Description:

Proper access control and role checking was ensured in all privileged functions.

Results:

All privileged functions performed proper access control.

5.2 REENTRANCY

Description:

Functions that sent ether or performed unsafe external calls were checked for reentrancy attacks.

Results:

No reentrancy attacks were possible. `transfer` and `refund` functions could lead to reentrancy, but were called after all status changes were performed, making the attack not worth.

5.3 SLIPPAGE

Description:

Slippage protection was ensured in the minting functions.

Results:

All the minting functions had slippage protection measures in place.

```
Running 16 tests for test/Halborn.t.sol:HalbornTest
[PASS] testFail_Roles2() (gas: 220)
[PASS] testFees() (gas: 187)
[PASS] testMintF_1() (gas: 188)
[PASS] testMintF_2() (gas: 165)
[PASS] testMintF_3() (gas: 166)
[PASS] testMintF_4() (gas: 167)
[PASS] testMintRedeem_1() (gas: 210)
[PASS] testMintRedeem_2() (gas: 186)
[PASS] testMintRedeem_3() (gas: 209)
[PASS] testMintX_1() (gas: 209)
[PASS] testMintX_2() (gas: 231)
[PASS] testMintX_3() (gas: 166)
[PASS] testRedeem() (gas: 208)
[PASS] testReentrantRefund() (gas: 165)
[PASS] testReentrantTransfer() (gas: 188)
[PASS] testRoles() (gas: 232)
Test result: ok. 16 passed; 0 failed; 0 skipped; finished in 475.04µs
```

AUTOMATED TESTING

6.1 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:

Slither results for slitherResults	
Finding	**Impact**
MoriGateway._transfer(address,uint256,address) (src/MoriGateway.sol#274-285) sends eth to arbitrary user Dangerous calls: - (_success) = _recipient.call{value: _amount}() (src/MoriGateway.sol#280)	High
MoriGateway.mintETHCByETH(uint256) (src/MoriGateway.sol#115-127) ignores return value by ILido(steth).submit{value: msg.value}(address(0)) (src/MoriGateway.sol#118)	Medium
MoriGateway.mintETHSByETH(uint256) (src/MoriGateway.sol#80-92) ignores return value by ILido(steth).submit{value: msg.value}(address(0)) (src/MoriGateway.sol#83)	Medium
ChainlinkTwapOracleV3.getTwap(uint256) (src/ChainlinkTwapOracleV3.sol#73-82) performs a multiplication on the result of a division: - twap = _ownerUpdatedPrices[(timestamp / EPOCH) * EPOCH] (src/ChainlinkTwapOracleV3.sol#79)	Medium

Finding	Impact
ChainlinkTwapOracleV3.getLatest() (src/ChainlinkTwapOracleV3.sol#60-69) ignores return value by (answer,updatedAt) = AggregatorV3Interface(chainlinkAggregator).latestRoundData() (src/ChainlinkTwapOracleV3.sol#61-63)	Medium
ChainlinkTwapOracleV3.findLastRoundBefore(uint256) (src/ChainlinkTwapOracleV3.sol#86-134) ignores return value by (roundID,answer,None,updatedAt,None) = AggregatorV3Interface(chainlinkAggregator).latestRoundData() (src/ChainlinkTwapOracleV3.sol#89-91)	Medium
Treasury._computeMultiple(uint256) (src/Treasury.sol#486-497) performs a multiplication on the result of a division: - _ratio = int256(_newPrice).sub(_lastPermissionedPrice).mul(PRECISION_I256).div(_lastPermissionedPrice) (src/Treasury.sol#491-494) - _fMultiple = _ratio.mul(int256(beta)).div(PRECISION_I256) (src/Treasury.sol#496)	Medium
Treasury.mint(uint256,address,ITreasury.MintOption) (src/Treasury.sol#263-301) performs a multiplication on the result of a division: - _totalVal = _baseIn.mul(_state.baseNav).div(PRECISION) (src/Treasury.sol#281) - _fTokenOut = _totalVal.mul(initialMintRatio).div(PRECISION) (src/Treasury.sol#282)	Medium
Reentrancy in Treasury.protocolSettle() (src/Treasury.sol#327-338): External calls: - _fNav = IFractionalToken(fToken).updateNav(_fMultiple) (src/Treasury.sol#333) State variables written after the call(s): - lastPermissionedPrice = _newPrice (src/Treasury.sol#337) Treasury.lastPermissionedPrice (src/Treasury.sol#91) can be used in cross function reentrancies: - Treasury._computeMultiple(uint256) (src/Treasury.sol#486-497) - Treasury.initializePrice() (src/Treasury.sol#358-367) - Treasury.lastPermissionedPrice (src/Treasury.sol#91) - Treasury.protocolSettle() (src/Treasury.sol#327-338)	Medium

Finding	Impact
Reentrancy in Treasury.redeem(uint256,uint256,address) (src/Treasury.sol#304-324): External calls: - IFractionalToken(fToken).burn(_owner,_fTokenIn) (src/Treasury.sol#314) - ILeveragedToken(xToken).burn(_owner,_xTokenIn) (src/Treasury.sol#318) State variables written after the call(s): - totalBaseToken = _state.baseSupply.sub(_baseOut) (src/Treasury.sol#321) Treasury.totalBaseToken (src/Treasury.sol#94) can be used in cross function reentrancies: - Treasury._loadSwapState() (src/Treasury.sol#442-473) - Treasury.mint(uint256,address,ITreasury.MintOption) (src/Treasury.sol#263-301) - Treasury.protocolSettle() (src/Treasury.sol#327-338) - Treasury.redeem(uint256,uint256,address) (src/Treasury.sol#304-324) - Treasury.totalBaseToken (src/Treasury.sol#94)	Medium
Market._defuctMintFee(uint256,uint256,uint256,uint256) (src/Market.sol#522-544) performs a multiplication on the result of a division: - _maxBaseIn = _maxBaseInBeforeSystemStabilityMode.mul(PRECISION).div(PRECISION - _feeRatio0) (src/Market.sol#528) - _fee = _maxBaseIn.mul(_feeRatio0).div(PRECISION) (src/Market.sol#535)	Medium
Market.redeem(uint256,uint256,address,uint256) (src/Market.sol#300-357) uses a dangerous strict equality: - require(bool,string)(_fTokenIn == 0 || _xTokenIn == 0,only redeem single side) (src/Market.sol#315)	Medium
Market.mintXToken(uint256,address,uint256) (src/Market.sol#274-297) ignores return value by (_maxBaseInBeforeSystemStabilityMode) = _treasury.maxMintableXToken(marketConfig.stabilityRatio) (src/Market.sol#288)	Medium
Market.redeem(uint256,uint256,address,uint256) (src/Market.sol#300-357) ignores return value by (_maxFTokenInBeforeSystemStabilityMode) = _treasury.maxRedeemableFToken(_marketConfig.stabilityRatio) (src/Market.sol#322)	Medium

Finding	Impact
Market.mintFToken(uint256,address,uint256) (src/Market.sol#238-271) ignores return value by (_fTokenMinted,None) = _treasury.mint(_amountWithoutFee,_recipient,ITreasury.MintOption.FToken) (src/Market.sol#267)	Medium
Market.mintXToken(uint256,address,uint256) (src/Market.sol#274-297) ignores return value by (None,_xTokenMinted) = _treasury.mint(_amountWithoutFee,_recipient,ITreasury.MintOption.XToken) (src/Market.sol#293)	Medium
Market.mintFToken(uint256,address,uint256) (src/Market.sol#238-271) ignores return value by (_maxBaseInBeforeSystemStabilityMode) = _treasury.maxMintableFToken(marketConfig.stabilityRatio) (src/Market.sol#252)	Medium
Market.redeem(uint256,uint256,address,uint256) (src/Market.sol#300-357) ignores return value by (_maxXTokenInBeforeSystemStabilityMode) = _treasury.maxRedeemableXToken(_marketConfig.stabilityRatio) (src/Market.sol#325)	Medium
MoriGateway.updateSlippage(uint256) (src/MoriGateway.sol#190-192) should emit an event for: - slippage = _slippage (src/MoriGateway.sol#191)	Low
MoriGateway.initialize(address,address,address,address,address,uint256)._xToken (src/MoriGateway.sol#54) lacks a zero-check on : - xToken = _xToken (src/MoriGateway.sol#62)	Low
MoriGateway.updatePool(address)._pool (src/MoriGateway.sol#184) lacks a zero-check on : - pool = _pool (src/MoriGateway.sol#185)	Low
MoriGateway.initialize(address,address,address,address,address,uint256)._fToken (src/MoriGateway.sol#53) lacks a zero-check on : - fToken = _fToken (src/MoriGateway.sol#61)	Low
MoriGateway.initialize(address,address,address,address,address,uint256)._steth (src/MoriGateway.sol#52) lacks a zero-check on : - steth = _steth (src/MoriGateway.sol#60)	Low
MoriGateway.initialize(address,address,address,address,address,uint256)._market (src/MoriGateway.sol#51) lacks a zero-check on : - market = _market (src/MoriGateway.sol#59)	Low
MoriGateway.initialize(address,address,address,address,address,uint256)._pool (src/MoriGateway.sol#55) lacks a zero-check on : - pool = _pool (src/MoriGateway.sol#63)	Low

Finding	Impact
ChainlinkTwapOracleV3.constructor(address,uint256,uint256,string).chainlinkAggregator_ (src/ChainlinkTwapOracleV3.sol#43) lacks a zero-check on : - chainlinkAggregator = chainlinkAggregator_ (src/ChainlinkTwapOracleV3.sol#48)	Low
ChainlinkTwapOracleV3._getChainlinkRoundData(uint80) (src/ChainlinkTwapOracleV3.sol#194-213) has external calls inside a loop: (success,returnData) = chainlinkAggregator.staticcall(abi.encodePacked(AggregatorV3Interface.getRoundData.selector,abi.encode(roundID))) (src/ChainlinkTwapOracleV3.sol#197-203)	Low
ChainlinkTwapOracleV3.updateTwapFromOwner(uint256,uint256) (src/ChainlinkTwapOracleV3.sol#217-236) uses timestamp for comparisons Dangerous comparisons: - require(bool,string)(timestamp <= block.timestamp - EPOCH * 2,Not ready for owner) (src/ChainlinkTwapOracleV3.sol#222-225)	Low
ChainlinkTwapOracleV3.getLatest() (src/ChainlinkTwapOracleV3.sol#60-69) uses timestamp for comparisons Dangerous comparisons: - require(bool,string)(updatedAt >= block.timestamp - chainlinkMessageExpiration,Stale price oracle) (src/ChainlinkTwapOracleV3.sol#64-67)	Low
ChainlinkTwapOracleV3._getTwapFromChainlink(uint256) (src/ChainlinkTwapOracleV3.sol#139-191) uses timestamp for comparisons Dangerous comparisons: - require(bool,string)(block.timestamp >= timestamp,Too soon) (src/ChainlinkTwapOracleV3.sol#142)	Low
FractionalToken.initialize(address,string,string)._treasury (src/FractionalToken.sol#53) lacks a zero-check on : - treasury = _treasury (src/FractionalToken.sol#59)	Low
LeveragedToken.initialize(address,address,string,string)._fToken (src/LeveragedToken.sol#47) lacks a zero-check on : - fToken = _fToken (src/LeveragedToken.sol#54)	Low
LeveragedToken.initialize(address,address,string,string)._treasury (src/LeveragedToken.sol#46) lacks a zero-check on : - treasury = _treasury (src/LeveragedToken.sol#53)	Low
Treasury.transferToStrategy(uint256) (src/Treasury.sol#341-346) should emit an event for: - strategyUnderlying += _amount (src/Treasury.sol#345)	Low

Finding	Impact
Treasury.mint(uint256,address,ITreasury.MintOption) (src/Treasury.sol#263-301) should emit an event for: - totalBaseToken = _state.baseSupply + _baseIn (src/Treasury.sol#293)	Low
Treasury.redeem(uint256,uint256,address) (src/Treasury.sol#304-324) should emit an event for: - totalBaseToken = _state.baseSupply.sub(_baseOut) (src/Treasury.sol#321)	Low
Treasury.updateStrategy(address)._strategy (src/Treasury.sol#371) lacks a zero-check on : - strategy = _strategy (src/Treasury.sol#372)	Low
Treasury.initialize(address,address,address,address,address,uint256,uint256)._priceOracle (src/Treasury.sol#139) lacks a zero-check on : - priceOracle = _priceOracle (src/Treasury.sol#149)	Low
Treasury.updatePriceOracle(address)._priceOracle (src/Treasury.sol#387) lacks a zero-check on : - priceOracle = _priceOracle (src/Treasury.sol#388)	Low
Treasury.initialize(address,address,address,address,address,uint256,uint256)._fToken (src/Treasury.sol#137) lacks a zero-check on : - fToken = _fToken (src/Treasury.sol#147)	Low
Treasury.initialize(address,address,address,address,address,uint256,uint256)._baseToken (src/Treasury.sol#136) lacks a zero-check on : - baseToken = _baseToken (src/Treasury.sol#146)	Low
Treasury.initialize(address,address,address,address,address,uint256,uint256)._market (src/Treasury.sol#135) lacks a zero-check on : - market = _market (src/Treasury.sol#145)	Low
Treasury.initialize(address,address,address,address,address,uint256,uint256)._xToken (src/Treasury.sol#138) lacks a zero-check on : - xToken = _xToken (src/Treasury.sol#148)	Low
Reentrancy in Treasury._transferBaseToken(uint256,address) (src/Treasury.sol#420-438): External calls: - IAssetStrategy(strategy).withdrawToTreasury(_diff) (src/Treasury.sol#427) State variables written after the call(s): - strategyUnderlying = strategyUnderlying.sub(_diff) (src/Treasury.sol#428)	Low

Finding	Impact
Reentrancy in Treasury.transferToStrategy(uint256) (src/Treasury.sol#341-346): External calls: - IERC20Upgradeable(baseToken).safeTransfer(strategy,_amount) (src/Treasury.sol#344) State variables written after the call(s): - strategyUnderlying += _amount (src/Treasury.sol#345)	Low
Reentrancy in Treasury.protocolSettle() (src/Treasury.sol#327-338): External calls: - _fNav = IFractionalToken(fToken).updateNav(_fMultiple) (src/Treasury.sol#333) Event emitted after the call(s): - ProtocolSettle(_newPrice,_fNav) (src/Treasury.sol#335)	Low
Reentrancy in Treasury.initializePrice() (src/Treasury.sol#358-367): External calls: - IFractionalToken(fToken).setNav(PRECISION) (src/Treasury.sol#364) Event emitted after the call(s): - ProtocolSettle(_price,PRECISION) (src/Treasury.sol#366)	Low
Treasury._fetchTwapPrice() (src/Treasury.sol#502-506) uses timestamp for comparisons Dangerous comparisons: - require(bool,string)(_price > 0,invalid twap price) (src/Treasury.sol#505)	Low
Market.updatePlatform(address)._platform (src/Market.sol#433) lacks a zero-check on : - platform = _platform (src/Market.sol#434)	Low
Market.initialize(address,address)._treasury (src/Market.sol#195) lacks a zero-check on : - treasury = _treasury (src/Market.sol#200) - baseToken = ITreasury(_treasury).baseToken() (src/Market.sol#203) - fToken = ITreasury(_treasury).fToken() (src/Market.sol#204) - xToken = ITreasury(_treasury).xToken() (src/Market.sol#205)	Low
Market.initialize(address,address)._platform (src/Market.sol#195) lacks a zero-check on : - platform = _platform (src/Market.sol#201)	Low
End of table for slitherResults	

Results summary:

The findings obtained as a result of the Slither scan were reviewed. The majority of Slither findings were determined false-positives.

6.2 AUTOMATED SECURITY SCAN

Description:

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

AUTOMATED TESTING

Results:

Report for src/ChainlinkTwapOracleV3.sol
https://dashboard.mythx.io/#/console/analyses/f553ae7f-28ae-4104-8bc5-ed8d39f3c323

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
54	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "**" discovered
65	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
79	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
79	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
92	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
97	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
99	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "--" discovered
109	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
117	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
118	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
118	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
156	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
157	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
159	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
164	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "--" discovered
175	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
181	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
185	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
187	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
221	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "%" discovered
223	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
223	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered

Report for src/FractionalToken.sol
https://dashboard.mythx.io/#/console/analyses/41fc1aec-eed9-477c-8512-727953ff40ff

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
79	(SWC-101) Integer Overflow and Underflow	High	The arithmetic operation can overflow.

Report for src/LeveragedToken.sol
https://dashboard.mythx.io/#/console/analyses/7a7d6315-45d8-4bcf-ad7d-b225da11b6ae

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Report for src/Treasury.sol
https://dashboard.mythx.io/#/console/analyses/3d7dd7a8-cec3-45a7-a8a4-4508c3ad270b

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
164	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
290	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
293	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
345	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
426	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered

Report for src/Market.sol
https://dashboard.mythx.io/#/console/analyses/44e06248-0e99-4f9f-a293-0b7d3362e4e3

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
16	(SWC-123) Requirement Violation	Low	Requirement violation.
226	(SWC-123) Requirement Violation	Low	Requirement violation.

The findings obtained as a result of the MythX scan were examined, and they were not included in the report because they were determined false positives.

THANK YOU FOR CHOOSING

// HALBORN