// HALBORN

JARVIS

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 09, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/09/2021	Nishit Majithia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Nishit Majithia	Halborn	nishit.majithia@halborn.com

EXECUTIVE SUMMARY

1.1 INTRODUCTION

JARVIS engaged Halborn to conduct a security assessment on their all smart contracts including Oracle and Synthereum (the protocol behind Jarvis Exchange). The security assessment was scoped to the smart contracts under these folders contracts/base/, contracts/derivative/, contracts/oracle/, contracts/synthereum-pool/v3/ and contracts/versioning/. An audit of the security risk and implications regarding the changes introduced by the development team at JARVIS prior to its production release shortly following the assessments deadline.

Overall, the smart contract code is well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted.

Though the outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verication of essential properties were performed to achieve objectives and deliverables set in the scope. It is important to remark the use of the best practices for secure smart contract development. Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 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 the smart contract audit.
While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow

security best practices. The following phases and associated tools were used throughout the term of the audit:
- Research into architecture and purpose
- Smart Contract manual code read 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.
- Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Truffle, Ganache, Infura)
- Smart Contract Fuzzing and dynamic state exploitation (Echidna) Symbolic Execution / EVM bytecode security assessment (limited time)

1.3 SCOPE

IN-SCOPE:
- The security assessment was scoped to the smart contracts under these folders contracts/base/, contracts/derivative/, contracts/oracle/, contracts/synthereum-pool/v3/ and contracts/versioning/.
Specific commit of contract: commit 27f7c257038b349352a0bdfcafba4a1c35fd569d

OUT-OF-SCOPE:
Other smart contracts in the repository, customized uma-fork, external libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
FLOATING PRAGMA	Low	-
MISSING ADDRESS CHECK	Low	-
USE OF BLOCK.TIMESTAMP	Informational	-
IGNORE RETURN VALUES	Informational	-
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN	-	-

FINDINGS & TECH DETAILS

3.1 FLOATING PRAGMA - LOW

Description:

All Smart Contracts use the floating pragma ^0.6.12. 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 updated compiler version that might introduce bugs or discovered vulnerabilities in the newest versions that affect the contract system negatively.

Code Location:

Every contract under the scope is using floating pragma.

Recommendations:

Consider lock the pragma version known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Pragma can also be locked fixing the compiler version in the configuration file when you deploy contracts with truffle or hardhat frameworks.

3.2 MISSING ADDRESS CHECK - LOW

Description:

Address validation at some places in contracts contracts/derivative/v1/DerivativeFactory.sol, contracts/derivative/v1/SyntheticTokenFactory.sol and contracts/synthereum-pool/v3/PoolChainPriceFeedFactory.sol. Lack of zero address validation has been found at many instances when assigning user supplied address values to state variables directly.

Code Location:

contracts/derivative/v1/DerivativeFactory.sol:

Line #30

```
21    constructor(
22      address _synthereumFinder,
23      address _umaFinder,
24      address _tokenFactoryAddress,
25      address _timerAddress
26    )
27      public
28      PerpetualPoolPartyCreator(_umaFinder, _tokenFactoryAddress, _timerAddress)
29    {
30      synthereumFinder = _synthereumFinder;
31      deploymentSignature = this.createPerpetual.selector;
32    }
```

contracts/derivative/v1/SyntheticTokenFactory.sol:

Line #22

```
20
21    constructor(address _synthereumFinder, uint8 _derivativeVersion) public {
22      synthereumFinder = _synthereumFinder;
23      derivativeVersion = _derivativeVersion;
24    }
25
26    function createToken(
```

contracts/synthereum-pool/v3/PoolChainPriceFeedFactory.sol:

Line #38

```
33    /**
34     * @notice Set synthereum finder
35     * @param _synthereumFinder Synthereum finder contract
36     */
37    constructor(address _synthereumFinder) public {
38      synthereumFinder = _synthereumFinder;
39      deploymentSignature = this.createPool.selector;
40    }
41
42    //----------------------------------------
43    // Public functions
```

Recommendation:

Add proper address validation when every state variable assignment done from user supplied input.

3.3 USE OF BLOCK.TIMESTAMP - INFORMATIONAL

Description:

Block timestamps have historically been used for a variety of applications, such as entropy for random numbers, locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

block.timestamp or it's alias now can be manipulated by miners if they have some incentive to do so

Code Location:

contracts/synthereum-pool/v3/PoolChainPriceFeedLib.sol:
Line #1134

```
1128    function checkParams(
1129      ISynthereumPoolOnChainPriceFeedStorage.Storage storage self,
1130      IDerivative derivative,
1131      uint256 feePercentage,
1132      uint256 expiration
1133    ) internal view checkDerivative(self, derivative) {
1134      require(now <= expiration, 'Transaction expired');
1135      require(
1136        self.fee.feePercentage.rawValue <= feePercentage,
1137        'User fee percentage less than actual one'
1138      );
1139    }
```

Recommendation:

Avoid getting relied on use of block.timestamp in require methods in order to prevent possible miners manipulation.

3.4 IGNORE RETURN VALUES - INFORMATIONAL

Description:

The return value of an external call is not stored in a local or state variable. In contracts contracts/synthereum-pool/v3/PoolOnChainPriceFeedLib.sol, contracts/versioning/PoolRegister.sol and contracts/versioning/FactoryVersioning.sol, there are few instances where external methods are being called and return value(bool) are being ignored.

Code Location:

contracts/synthereum-pool/v3/PoolOnChainPriceFeedLib.sol:

Line #178

```
171     self.version = _version;
172     self.finder = _finder;
173     self.startingCollateralization = _startingCollateralization;
174     self.isContractAllowed = _isContractAllowed;
175     self.collateralToken = getDerivativeCollateral(_derivative);
176     self.syntheticToken = _derivative.tokenCurrency();
177     self.priceIdentifier = _derivative.positionManagerData().priceIdentifier;
178     self.derivatives.add(address(_derivative));
179     emit AddDerivative(address(this), address(_derivative));
180   }
181
```

Line #596

```
593     }
594
595     // Redeem the synthetic tokens for collateral
596     derivative.settleEmergencyShutdown();
597
598     // Amount of collateral that will be redeemed and sent to the user
599     FixedPoint.Unsigned memory totalToRedeem;
600
601     // If the user is the LP, send redeemed token collateral plus excess collateral
602     if (isLiquidityProvider) {
```

Line #719

```
717           finder.getImplementationAddress(SynthereumInterfaces.PoolRegistry)
718         );
719       poolRegister.isPoolDeployed(
720         pool.syntheticTokenSymbol(),
721         collateralToken,
722         pool.version(),
723         address(pool)
724       );
725       if (
726         derivativeRole == ISynthereumPoolOnChainPriceFeed.DerivativeRoles.POOL
727       ) {
```

contracts/versioning/PoolRegister.sol:

Line #40-41

```
36        ISynthereumFinder(synthereumFinder).getImplementationAddress(
37          SynthereumInterfaces.Deployer
38        );
39      require(msg.sender == deployer, 'Sender must be Synthereum deployer');
40      symbolToPools[syntheticTokenSymbol][collateralToken][poolVersion].add(pool);
41      collaterals.add(address(collateralToken));
42    }
43
44    function isPoolDeployed(
45      string calldata poolSymbol,
46      IERC20 collateral,
```

contracts/versioning/FactoryVersioning.sol

Line #66

Line #77

```
64    function removePoolFactory(uint8 version) external override onlyMaintainer {
65      require(
66        _poolsFactory.remove(version),
67        'Version of the pool factory does not exist'
68      );
69      emit RemovePoolFactory(version);
70    }
71
72    function setDerivativeFactory(uint8 version, address derivativeFactory)
73      external
74      override
75      onlyMaintainer
76    {
77      _derivativeFactory.set(version, derivativeFactory);
78      emit AddDerivativeFactory(version, derivativeFactory);
79    }
```

Recommendation:

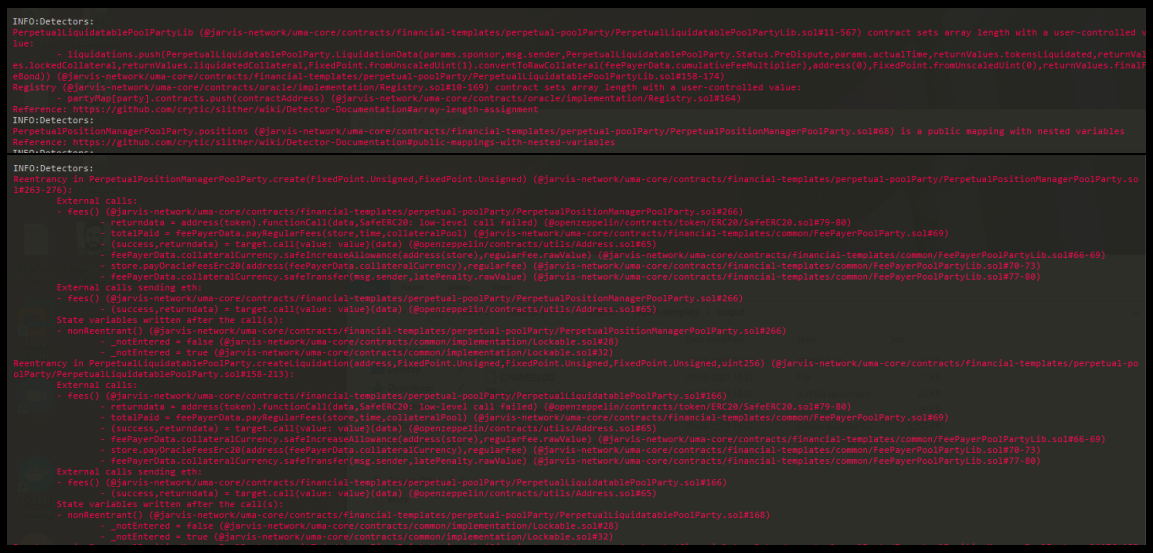
Add return value check to avoid unexpected crash of the contract. Return value check will help in handling the exceptions better way.

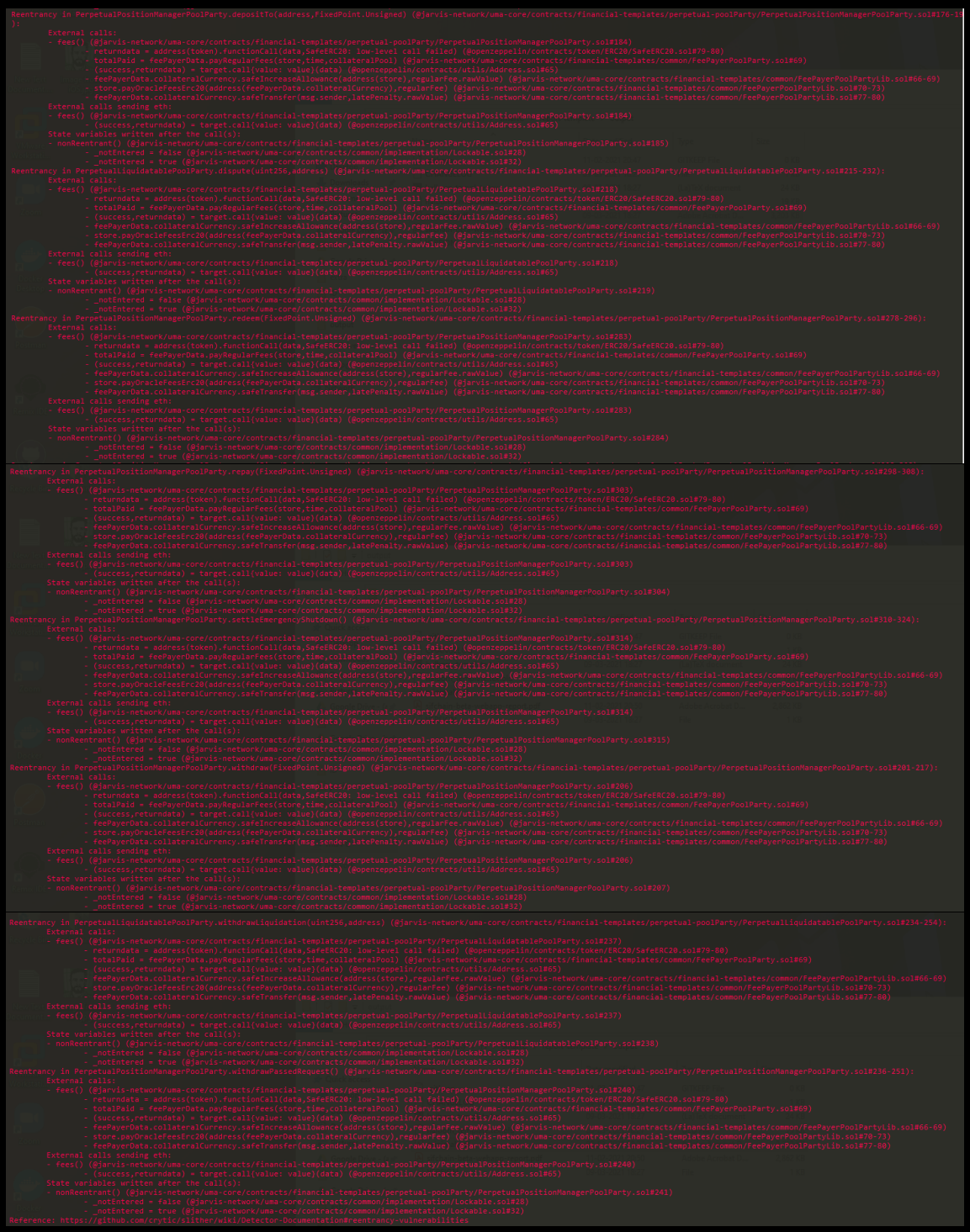
3.5 STATIC ANALYSIS REPORT

Description:

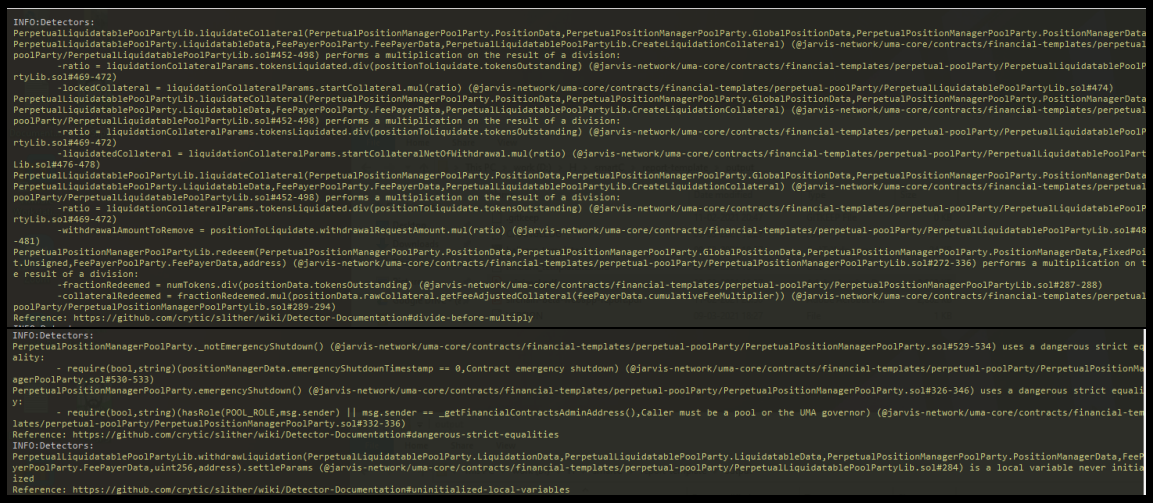
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats. 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.

Results:

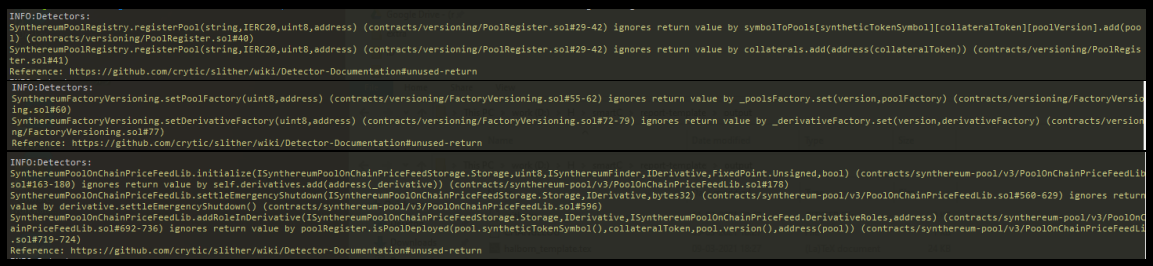




- These issues highlighted by slither is in Jarvis's UMA-fork or in openzeppelin library, which is out of the scope for this Audit. But this issue has been already covered in Halborn's security audit of Jarvis's UMA-Fork smart contracts.



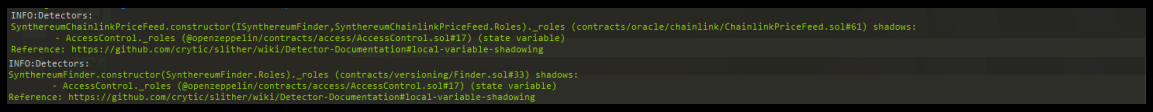
- Again above Divide Before Multiply and Dangerous Strict Equalities issues are in the Jarvis's UMA-Fork library and out of scope for this audit.



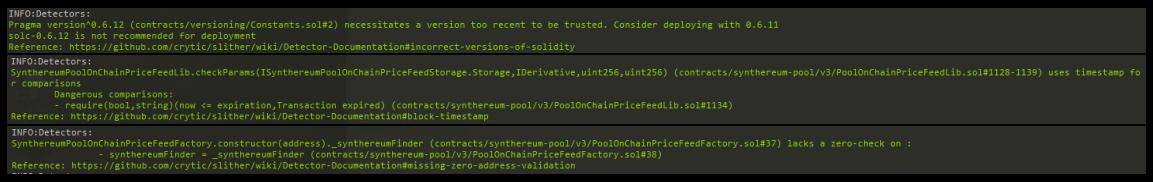
- These issues have already mentioned in the analysis report.

- Above issues are related to Jarvis's UMA-Fork, which were out of the scope for this audit.

- This missing zero-address validation issues have already mentioned in the analysis report.



- Local variable shadowing is not impacting here since AccesssControl._roles variable is not used by contract. So it is false positive.



- These issues have already mentioned in the analysis report.

3.6 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit 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 testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. In addition, Security Detections are only in scope.

Results:

contracts/synthereum-pool/v3/PoolOnChainPriceFeed.sol

Report for contracts/synthereum-pool/v3/PoolOnChainPriceFeed.sol
https://dashboard.mythx.io/#/console/analyses/345e4951-7571-4156-a131-c594314599c5

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

contracts/synthereum-pool/v3/PoolOnChainPriceFeedLib.sol

Report for contracts/synthereum-pool/v3/PoolOnChainPriceFeedLib.sol
https://dashboard.mythx.io/#/console/analyses/345e4951-7571-4156-a131-c594314599c5

Line	SWC Title	Severity	Short Description
141	(SWC-115) Authorization through tx.origin	Low	Use of "tx.origin" as a part of authorization control.
1128	(SWC-000) Unknown	Medium	Incorrect function "checkParams" state mutability

contracts/synthereum-pool/v3/PoolOnChainPriceFeedFactory.sol

Report for contracts/synthereum-pool/v3/PoolOnChainPriceFeedFactory.sol
https://dashboard.mythx.io/#/console/analyses/25351a22-87e8-4cb2-ba01-e354a2e020ad

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

contracts/synthereum-pool/v3/PoolOnChainPriceFeedCreator.sol

Report for contracts/synthereum-pool/v3/PoolOnChainPriceFeedCreator.sol
https://dashboard.mythx.io/#/console/analyses/52dcbcf8-cdeb-4e70-9b75-9f9c035b750b

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

contracts/versioning/Constants.sol

Report for contracts/versioning/Constants.sol
https://dashboard.mythx.io/#/console/analyses/27af861e-484f-4c28-add1-6bd69d4a8f1e

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

contracts/versioning/Deployer.sol

Report for contracts/versioning/Deployer.sol
https://dashboard.mythx.io/#/console/analyses/d0394098-2ddc-4e1a-bc87-570b1ee612b2

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

contracts/oracle/chainlink/ChainlinkPriceFeed.sol

Report for contracts/oracle/chainlink/ChainlinkPriceFeed.sol
https://dashboard.mythx.io/#/console/analyses/eebf8c69-c1fa-4eaa-82d2-ffcf4f59638c

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
82	(SWC-115) Authorization through tx.origin	Low	Use of "tx.origin" as a part of authorization control.

contracts/derivative/v1/SyntheticTokenFactory.sol

Report for contracts/derivative/v1/SyntheticTokenFactory.sol
https://dashboard.mythx.io/#/console/analyses/d21d2c1f-c0ff-4837-92a7-c62453fe482a

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
26	(SWC-000) Unknown	Medium	Function could be marked as external.

contracts/derivative/v1/DerivativeFactory.sol

Report for contracts/derivative/v1/DerivativeFactory.sol
https://dashboard.mythx.io/#/console/analyses/178e7679-99ce-49a1-97e7-8107b07151d6

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
34	(SWC-000) Unknown	Medium	Function could be marked as external.

THANK YOU FOR CHOOSING

// HALBORN