// HALBORN

MetaStreet Labs - MetaStreet Contracts

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 18th, 2022 - May 17th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/09/2022	István Böhm
0.2	Document Updates	05/11/2022	István Böhm
0.3	Draft Review	05/11/2022	Roberto Reigada
0.4	Final Draft Review	05/11/2022	Gabi Urrutia
1.0	Remediation Plan	05/18/2022	István Böhm
1.1	Remediation Plan Review	05/18/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
István Böhm	Halborn	Istvan.Bohm@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

MetaStreet Labs engaged Halborn to conduct a security audit on their smart contracts beginning on April 18th, 2022 and ending on May 17th, 2022. The security assessment was scoped to the smart contracts provided in the contracts GitHub repository metastreet-labs/metastreet-contracts.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned one full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified few security risks that were addressed by the MetaStreet Labs team.

1.3 TEST APPROACH & METHODOLOGY

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

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the following smart contracts:

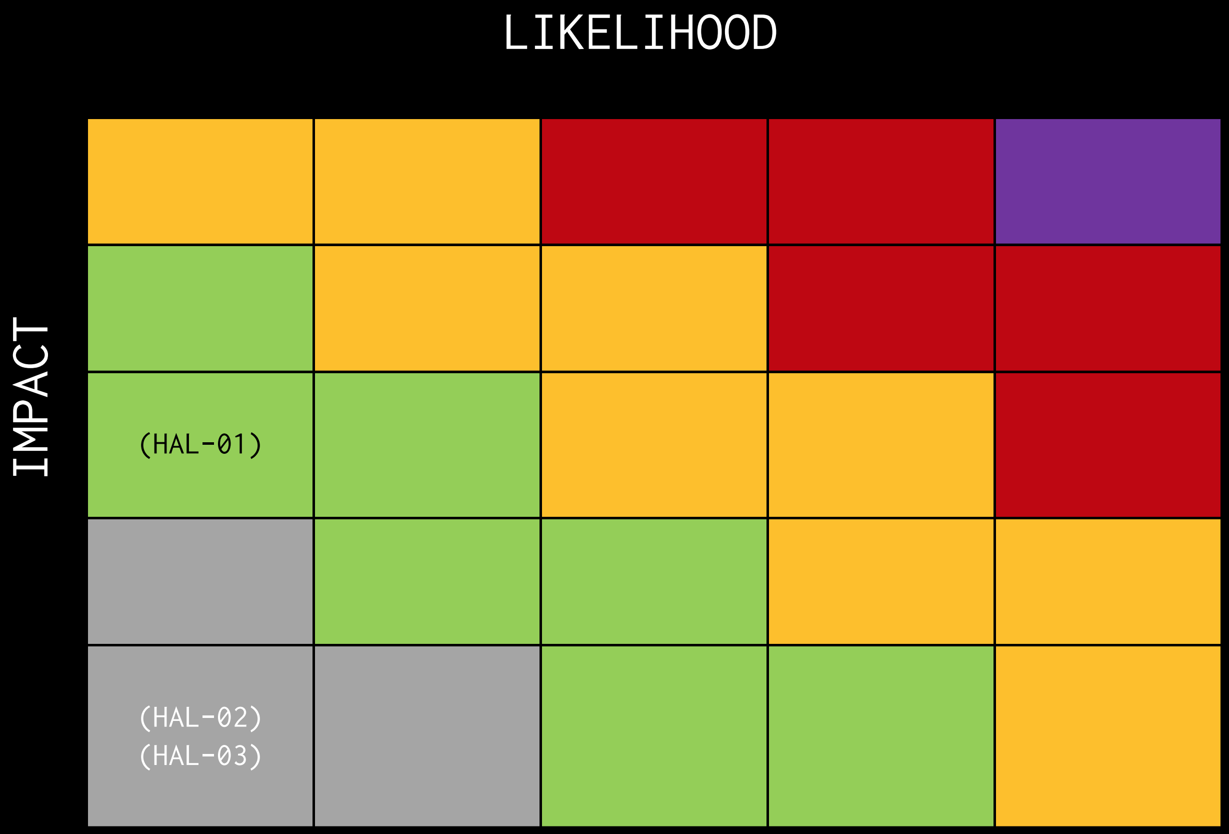
- integrations/ArcadeV1/ArcadeV1NoteAdapter.sol
- integrations/ArcadeV1/LoanLibrary.sol
- integrations/NFTfiV2/NFTfiV2NoteAdapter.sol
- interfaces/ILoanPriceOracle.sol
- interfaces/ILoanReceiver.sol
- interfaces/INoteAdapter.sol
- interfaces/IVault.sol
- interfaces/IVaultRegistry.sol
- LPToken.sol
- LoanPriceOracle.sol
- Vault.sol
- VaultRegistry.sol

Commit ID:
- 3c0775012ec9117c801c00ca530f6cb1f6da5151

Fixed Commit ID:
- 2d69c087a21c95de818043e9a3c3d5573b6140f1

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - MISSING RE-ENTRANCY PROTECTION	Low	SOLVED - 05/18/2022
HAL02 - UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0	Informational	SOLVED - 05/18/2022
HAL03 - USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS	Informational	SOLVED - 05/18/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING RE-ENTRANCY PROTECTION - LOW

Description:

The Vault contract missed nonReentrant guard on the withdraw, redeem, withdrawAdminFees, sellNoteAndDeposit, and onCollateralLiquidated external functions. Even if the functions follow the check-effects-interactions pattern, we recommend using a mutex to protect against cross-function re-entrancy attacks. By using this lock, an attacker can no longer exploit the function with a recursive call. OpenZeppelin has its own mutex implementation for upgradeable contracts called ReentrancyGuardUpgradeable which provides a modifier to any function called nonReentrant that guards the function with a mutex against the re-entrancy attacks.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

We recommend using ReentrancyGuardUpgradeable via the nonReentrant modifier.

Remediation Plan:

SOLVED: The MetaStreet Labs team added the nonReentrant modifier to the withdraw, redeem, withdrawAdminFees, sellNoteAndDeposit, and onCollateralLiquidated functions.

3.2 (HAL-02) UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0 - INFORMATIONAL

Description:

Since `i` is a `uint256`, it is already initialized to 0. `uint256 i = 0` reassigns the 0 to `i` which wastes gas.

Code Location:

Vault.sol
- Line 588: `for (uint256 i = 0; i < SHARE_PRICE_PRORATION_BUCKETS; i++) {`
- Line 1113: `for (uint256 i = 0; i < numNoteTokens; i++){`
- Line 1128: `for (uint256 j = 0; j < numLoans; j++){`

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not to initialize uint variables to 0 to save some gas. For example, use instead:
`for (uint256 i; i < SHARE_PRICE_PRORATION_BUCKETS; ++i){`.

Remediation Plan:

SOLVED: The MetaStreet Labs team removed the unnecessary initialization.

3.3 (HAL-03) USING ++I CONSUMES LESS GAS THAN I++ IN LOOPS - INFORMATIONAL

Description:

In the following loops, the i variable is incremented using i++. It is known that, in loops, using ++i costs less gas per iteration than i++.

Code Location:

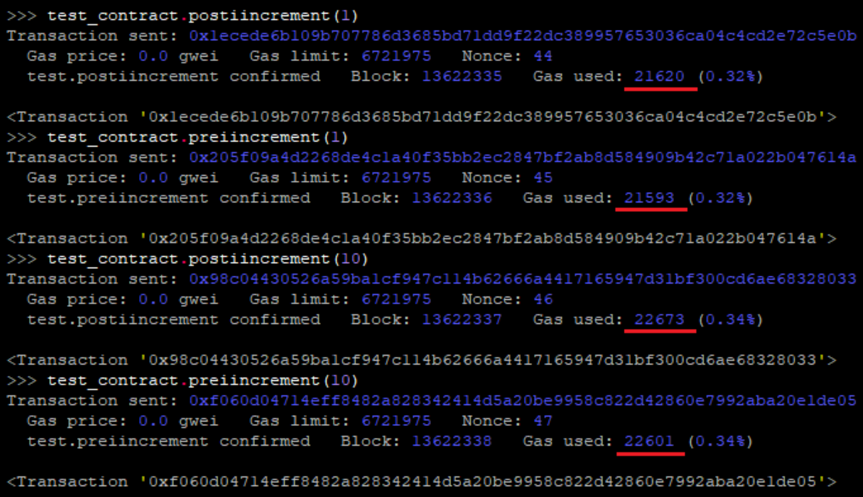
Vault.sol
- Line 588: for (uint256 i = 0; i < SHARE_PRICE_PRORATION_BUCKETS; i++) {
- Line 1113: for (uint256 i = 0; i < numNoteTokens; i++){
- Line 1121:
for (uint256 timeBucket = currentTimeBucket - 1; timeBucket < currentTimeBucket + SHARE_PRICE_PRORATION_BUCKETS; timeBucket++){
- Line 1128: for (uint256 j = 0; j < numLoans; j++){

Proof of Concept:

For example, based on the following test contract:

Listing 1: Test.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.9;

contract test {
    function postiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; i++) {
        }
    }
    function preiincrement(uint256 iterations) public {
        for (uint256 i = 0; i < iterations; ++i) {
        }
    }
}
```



```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59ba1cf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of a uint variable inside a loop. This does not just apply to the iterator variable. It also applies to increments made within the loop code block.

Remediation Plan:

SOLVED: The MetaStreet Labs team replaced postfix (i++) operators with prefix operators (++i) in loops.

AUTOMATED TESTING

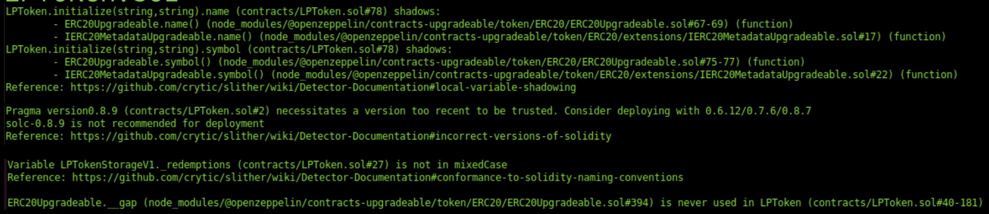
4.1 STATIC ANALYSIS REPORT

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.

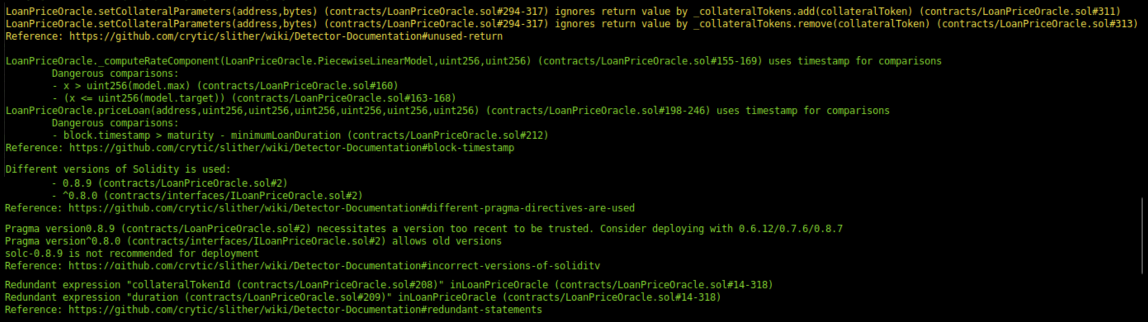
Slither Results:

LPToken.sol



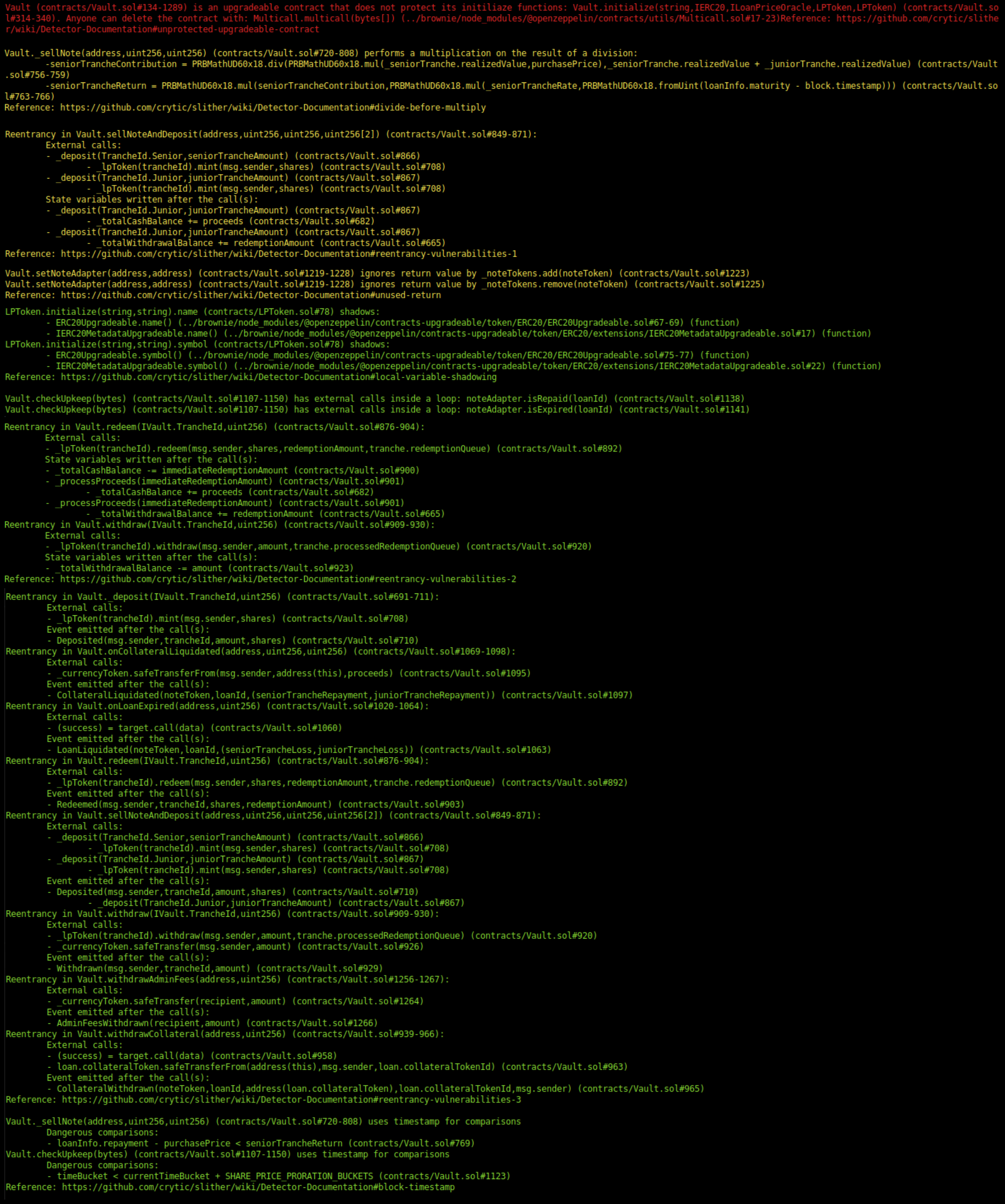
```
LPToken.initialize(string,string).name (contracts/LPToken.sol#78) shadows:
        - ERC20Upgradeable.name() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#67-69) (function)
        - IERC20MetadataUpgradeable.name() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#17) (function)
LPToken.initialize(string,string).symbol (contracts/LPToken.sol#78) shadows:
        - ERC20Upgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#75-77) (function)
        - IERC20MetadataUpgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#22) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Pragma version0.8.9 (contracts/LPToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable LPTokenStorageV1._redemptions (contracts/LPToken.sol#27) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

ERC20Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#394) is never used in LPToken (contracts/LPToken.sol#40-181)
```

LoanPriceOracle.sol



```
LoanPriceOracle.setCollateralParameters(address,bytes) (contracts/LoanPriceOracle.sol#294-317) ignores return value by _collateralTokens.add(collateralToken) (contracts/LoanPriceOracle.sol#311)
LoanPriceOracle.setCollateralParameters(address,bytes) (contracts/LoanPriceOracle.sol#294-317) ignores return value by _collateralTokens.remove(collateralToken) (contracts/LoanPriceOracle.sol#313)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

LoanPriceOracle._computeRateComponent(LoanPriceOracle.PiecewiseLinearModel,uint256,uint256) (contracts/LoanPriceOracle.sol#155-169) uses timestamp for comparisons
        Dangerous comparisons:
        - x > uint256(model.max) (contracts/LoanPriceOracle.sol#160)
        - (x <= uint256(model.target)) (contracts/LoanPriceOracle.sol#163-168)
LoanPriceOracle.priceLoan(address,uint256,uint256,uint256,uint256,uint256,uint256) (contracts/LoanPriceOracle.sol#198-246) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > maturity - minimumLoanDuration (contracts/LoanPriceOracle.sol#212)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Different versions of Solidity is used:
        - 0.8.9 (contracts/LoanPriceOracle.sol#2)
        - ^0.8.0 (contracts/interfaces/ILoanPriceOracle.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version0.8.9 (contracts/LoanPriceOracle.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (contracts/interfaces/ILoanPriceOracle.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Redundant expression "collateralTokenId (contracts/LoanPriceOracle.sol#208)" inLoanPriceOracle (contracts/LoanPriceOracle.sol#14-318)
Redundant expression "duration (contracts/LoanPriceOracle.sol#209)" inLoanPriceOracle (contracts/LoanPriceOracle.sol#14-318)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

Vault.sol



```
Vault (contracts/Vault.sol#134-1289) is an upgradeable contract that does not protect its initiliaze functions: Vault.initialize(string,IERC20,ILoanPriceOracle,LPToken,LPToken) (contracts/Vault.so
l#314-340). Anyone can delete the contract with: Multicall.multicall(bytes[]) (../brownie/node_modules/@openzeppelin/contracts/utils/Multicall.sol#17-23)Reference: https://github.com/crytic/slithe
r/wiki/Detector-Documentation#unprotected-upgradeable-contract

Vault._sellNote(address,uint256,uint256) (contracts/Vault.sol#720-808) performs a multiplication on the result of a division:
        -seniorTrancheContribution = PRBMathUD60x18.div(PRBMathUD60x18.mul(_seniorTranche.realizedValue,purchasePrice),_seniorTranche.realizedValue + _juniorTranche.realizedValue) (contracts/Vault
.sol#756-759)
        -seniorTrancheReturn = PRBMathUD60x18.mul(seniorTrancheContribution,PRBMathUD60x18.mul(_seniorTrancheRate,PRBMathUD60x18.fromUint(loanInfo.maturity - block.timestamp))) (contracts/Vault.so
l#763-766)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply


Reentrancy in Vault.sellNoteAndDeposit(address,uint256,uint256,uint256[2]) (contracts/Vault.sol#849-871):
        External calls:
        - _deposit(TrancheId.Senior,seniorTrancheAmount) (contracts/Vault.sol#866)
                - _lpToken(trancheId).mint(msg.sender,shares) (contracts/Vault.sol#708)
        - _deposit(TrancheId.Junior,juniorTrancheAmount) (contracts/Vault.sol#867)
                - _lpToken(trancheId).mint(msg.sender,shares) (contracts/Vault.sol#708)
        State variables written after the call(s):
        - _deposit(TrancheId.Junior,juniorTrancheAmount) (contracts/Vault.sol#867)
                - _totalCashBalance += proceeds (contracts/Vault.sol#682)
        - _deposit(TrancheId.Junior,juniorTrancheAmount) (contracts/Vault.sol#867)
                - _totalWithdrawalBalance += redemptionAmount (contracts/Vault.sol#665)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Vault.setNoteAdapter(address,address) (contracts/Vault.sol#1219-1228) ignores return value by _noteTokens.add(noteToken) (contracts/Vault.sol#1223)
Vault.setNoteAdapter(address,address) (contracts/Vault.sol#1219-1228) ignores return value by _noteTokens.remove(noteToken) (contracts/Vault.sol#1225)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

LPToken.initialize(string,string).name (contracts/LPToken.sol#78) shadows:
        - ERC20Upgradeable.name() (../brownie/node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#67-69) (function)
        - IERC20MetadataUpgradeable.name() (../brownie/node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#17) (function)
LPToken.initialize(string,string).symbol (contracts/LPToken.sol#78) shadows:
        - ERC20Upgradeable.symbol() (../brownie/node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#75-77) (function)
        - IERC20MetadataUpgradeable.symbol() (../brownie/node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#22) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Vault.checkUpkeep(bytes) (contracts/Vault.sol#1107-1150) has external calls inside a loop: noteAdapter.isRepaid(loanId) (contracts/Vault.sol#1138)
Vault.checkUpkeep(bytes) (contracts/Vault.sol#1107-1150) has external calls inside a loop: noteAdapter.isExpired(loanId) (contracts/Vault.sol#1141)

Reentrancy in Vault.redeem(IVault.TrancheId,uint256) (contracts/Vault.sol#876-904):
        External calls:
        - _lpToken(trancheId).redeem(msg.sender,shares,redemptionAmount,tranche.redemptionQueue) (contracts/Vault.sol#892)
        State variables written after the call(s):
        - _totalCashBalance -= immediateRedemptionAmount (contracts/Vault.sol#900)
        - _processProceeds(immediateRedemptionAmount) (contracts/Vault.sol#901)
                - _totalCashBalance += proceeds (contracts/Vault.sol#682)
        - _processProceeds(immediateRedemptionAmount) (contracts/Vault.sol#901)
                - _totalWithdrawalBalance += redemptionAmount (contracts/Vault.sol#665)
Reentrancy in Vault.withdraw(IVault.TrancheId,uint256) (contracts/Vault.sol#909-930):
        External calls:
        - _lpToken(trancheId).withdraw(msg.sender,amount,tranche.processedRedemptionQueue) (contracts/Vault.sol#920)
        State variables written after the call(s):
        - _totalWithdrawalBalance -= amount (contracts/Vault.sol#923)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in Vault._deposit(IVault.TrancheId,uint256) (contracts/Vault.sol#691-711):
        External calls:
        - _lpToken(trancheId).mint(msg.sender,shares) (contracts/Vault.sol#708)
        Event emitted after the call(s):
        - Deposited(msg.sender,trancheId,amount,shares) (contracts/Vault.sol#710)
Reentrancy in Vault.onCollateralLiquidated(address,uint256,uint256) (contracts/Vault.sol#1069-1098):
        External calls:
        - _currencyToken.safeTransferFrom(msg.sender,address(this),proceeds) (contracts/Vault.sol#1095)
        Event emitted after the call(s):
        - CollateralLiquidated(noteToken,loanId,(seniorTrancheRepayment,juniorTrancheRepayment)) (contracts/Vault.sol#1097)
Reentrancy in Vault.onLoanExpired(address,uint256) (contracts/Vault.sol#1020-1064):
        External calls:
        - (success) = target.call(data) (contracts/Vault.sol#1060)
        Event emitted after the call(s):
        - LoanLiquidated(noteToken,loanId,(seniorTrancheLoss,juniorTrancheLoss)) (contracts/Vault.sol#1063)
Reentrancy in Vault.redeem(IVault.TrancheId,uint256) (contracts/Vault.sol#876-904):
        External calls:
        - _lpToken(trancheId).redeem(msg.sender,shares,redemptionAmount,tranche.redemptionQueue) (contracts/Vault.sol#892)
        Event emitted after the call(s):
        - Redeemed(msg.sender,trancheId,shares,redemptionAmount) (contracts/Vault.sol#903)
Reentrancy in Vault.sellNoteAndDeposit(address,uint256,uint256,uint256[2]) (contracts/Vault.sol#849-871):
        External calls:
        - _deposit(TrancheId.Senior,seniorTrancheAmount) (contracts/Vault.sol#866)
                - _lpToken(trancheId).mint(msg.sender,shares) (contracts/Vault.sol#708)
        - _deposit(TrancheId.Junior,juniorTrancheAmount) (contracts/Vault.sol#867)
                - _lpToken(trancheId).mint(msg.sender,shares) (contracts/Vault.sol#708)
        Event emitted after the call(s):
        - Deposited(msg.sender,trancheId,amount,shares) (contracts/Vault.sol#710)
                - _deposit(TrancheId.Junior,juniorTrancheAmount) (contracts/Vault.sol#867)
Reentrancy in Vault.withdraw(IVault.TrancheId,uint256) (contracts/Vault.sol#909-930):
        External calls:
        - _lpToken(trancheId).withdraw(msg.sender,amount,tranche.processedRedemptionQueue) (contracts/Vault.sol#920)
        - _currencyToken.safeTransfer(msg.sender,amount) (contracts/Vault.sol#926)
        Event emitted after the call(s):
        - Withdrawn(msg.sender,trancheId,amount) (contracts/Vault.sol#929)
Reentrancy in Vault.withdrawAdminFees(address,uint256) (contracts/Vault.sol#1256-1267):
        External calls:
        - _currencyToken.safeTransfer(recipient,amount) (contracts/Vault.sol#1264)
        Event emitted after the call(s):
        - AdminFeesWithdrawn(recipient,amount) (contracts/Vault.sol#1266)
Reentrancy in Vault.withdrawCollateral(address,uint256) (contracts/Vault.sol#939-966):
        External calls:
        - (success) = target.call(data) (contracts/Vault.sol#958)
        - loan.collateralToken.safeTransferFrom(address(this),msg.sender,loan.collateralTokenId) (contracts/Vault.sol#963)
        Event emitted after the call(s):
        - CollateralWithdrawn(noteToken,loanId,address(loan.collateralToken),loan.collateralTokenId,msg.sender) (contracts/Vault.sol#965)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Vault._sellNote(address,uint256,uint256) (contracts/Vault.sol#720-808) uses timestamp for comparisons
        Dangerous comparisons:
        - loanInfo.repayment - purchasePrice < seniorTrancheReturn (contracts/Vault.sol#769)
Vault.checkUpkeep(bytes) (contracts/Vault.sol#1107-1150) uses timestamp for comparisons
        Dangerous comparisons:
        - timeBucket < currentTimeBucket + SHARE_PRICE_PRORATION_BUCKETS (contracts/Vault.sol#1123)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Different versions of Solidity is used:
        - 0.8.9 (contracts/LPToken.sol#2)
        - 0.8.9 (contracts/Vault.sol#2)
        - ^0.8.0 (contracts/interfaces/ILoanPriceOracle.sol#2)
        - ^0.8.0 (contracts/interfaces/ILoanReceiver.sol#2)
        - ^0.8.0 (contracts/interfaces/INoteAdapter.sol#2)
        - ^0.8.0 (contracts/interfaces/IVault.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version0.8.9 (contracts/LPToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/Vault.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (contracts/interfaces/ILoanPriceOracle.sol#2) allows old versions
Pragma version^0.8.0 (contracts/interfaces/ILoanReceiver.sol#2) allows old versions
Pragma version^0.8.0 (contracts/interfaces/INoteAdapter.sol#2) allows old versions
Pragma version^0.8.0 (contracts/interfaces/IVault.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Vault.withdrawCollateral(address,uint256) (contracts/Vault.sol#939-966):
        - (success) = target.call(data) (contracts/Vault.sol#958)
Low level call in Vault.onLoanExpired(address,uint256) (contracts/Vault.sol#1020-1064):
        - (success) = target.call(data) (contracts/Vault.sol#1060)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable LPTokenStorageV1._redemptions (contracts/LPToken.sol#27) is not in mixedCase
Variable VaultStorageV1._name (contracts/Vault.sol#80) is not in mixedCase
Variable VaultStorageV1._currencyToken (contracts/Vault.sol#81) is not in mixedCase
Variable VaultStorageV1._loanPriceOracle (contracts/Vault.sol#82) is not in mixedCase
Variable VaultStorageV1._noteAdapters (contracts/Vault.sol#83) is not in mixedCase
Variable VaultStorageV1._noteTokens (contracts/Vault.sol#84) is not in mixedCase
Variable VaultStorageV1._seniorLPToken (contracts/Vault.sol#85) is not in mixedCase
Variable VaultStorageV1._juniorLPToken (contracts/Vault.sol#86) is not in mixedCase
Variable VaultStorageV1._seniorTrancheRate (contracts/Vault.sol#95) is not in mixedCase
Variable VaultStorageV1._adminFeeRate (contracts/Vault.sol#100) is not in mixedCase
Variable VaultStorageV1._seniorTranche (contracts/Vault.sol#106) is not in mixedCase
Variable VaultStorageV1._juniorTranche (contracts/Vault.sol#107) is not in mixedCase
Variable VaultStorageV1._totalCashBalance (contracts/Vault.sol#108) is not in mixedCase
Variable VaultStorageV1._totalAdminFeeBalance (contracts/Vault.sol#110) is not in mixedCase
Variable VaultStorageV1._totalWithdrawalBalance (contracts/Vault.sol#111) is not in mixedCase
Variable VaultStorageV1._loans (contracts/Vault.sol#116) is not in mixedCase
Variable VaultStorageV1._pendingLoans (contracts/Vault.sol#121) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable VaultStorageV1._currencyToken (contracts/Vault.sol#81) is too similar to Vault.initialize(string,IERC20,ILoanPriceOracle,LPToken,LPToken).currencyToken_ (contracts/Vault.sol#316)
Variable VaultStorageV1._juniorLPToken (contracts/Vault.sol#86) is too similar to Vault.initialize(string,IERC20,ILoanPriceOracle,LPToken,LPToken).juniorLPToken_ (contracts/Vault.sol#319)
Variable VaultStorageV1._loanPriceOracle (contracts/Vault.sol#82) is too similar to Vault.setLoanPriceOracle(address).loanPriceOracle_ (contracts/Vault.sol#1205)
Variable VaultStorageV1._loanPriceOracle (contracts/Vault.sol#82) is too similar to Vault.initialize(string,IERC20,ILoanPriceOracle,LPToken,LPToken).loanPriceOracle_ (contracts/Vault.sol#317)
Variable VaultStorageV1._seniorLPToken (contracts/Vault.sol#85) is too similar to Vault.initialize(string,IERC20,ILoanPriceOracle,LPToken,LPToken).seniorLPToken_ (contracts/Vault.sol#318)
Variable Vault.onCollateralLiquidated(address,uint256,uint256).juniorTrancheRepayment (contracts/Vault.sol#1082) is too similar to Vault.onCollateralLiquidated(address,uint256,uint256).seniorTranc
heRepayment (contracts/Vault.sol#1081)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

ERC20Upgradeable.__gap (../brownie/node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#394) is never used in LPToken (contracts/LPToken.sol#40-181)
ReentrancyGuardUpgradeable.__gap (../brownie/node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol#74) is never used in Vault (contracts/Vault.sol#134-1289)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

VaultRegistry.sol

```
Different versions of Solidity is used:
        - 0.8.9 (contracts/VaultRegistry.sol#2)
        - ^0.8.0 (contracts/interfaces/IVaultRegistry.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version0.8.9 (contracts/VaultRegistry.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (contracts/interfaces/IVaultRegistry.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

ArcadeV1NoteAdapter.sol

```
Reentrancy in ArcadeV1NoteAdapter.constructor(ILoanCore) (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#86-91):
        External calls:
        - _borrowerNote = loanCore.borrowerNote() (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#88)
        - _lenderNote = loanCore.lenderNote() (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#89)
        State variables written after the call(s):
        - _lenderNote = loanCore.lenderNote() (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#89)
Reentrancy in ArcadeV1NoteAdapter.constructor(ILoanCore) (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#86-91):
        External calls:
        - _borrowerNote = loanCore.borrowerNote() (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#88)
        - _lenderNote = loanCore.lenderNote() (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#89)
        - _collateralToken = IAssetWrapper(address(loanCore.collateralToken())) (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#90)
        State variables written after the call(s):
        - _collateralToken = IAssetWrapper(address(loanCore.collateralToken())) (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#90)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

ArcadeV1NoteAdapter.isExpired(uint256) (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#202-207) uses timestamp for comparisons
        Dangerous comparisons:
        - loanData.state == LoanLibrary.LoanState.Active && block.timestamp > loanData.dueDate (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#206)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Different versions of Solidity is used:
        - 0.8.9 (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#2)
        - ^0.8.0 (contracts/integrations/ArcadeV1/LoanLibrary.sol#2)
        - ^0.8.0 (contracts/interfaces/INoteAdapter.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version0.8.9 (contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (contracts/integrations/ArcadeV1/LoanLibrary.sol#2) allows old versions
Pragma version^0.8.0 (contracts/interfaces/INoteAdapter.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

NFTfiV2NoteAdapter.sol

```
NFTfiV2NoteAdapter.isExpired(uint256) (contracts/integrations/NFTfiV2/NFTfiV2NoteAdapter.sol#232-243) uses timestamp for comparisons
        Dangerous comparisons:
        - loanData.status == IDirectLoanCoordinator.StatusType.NEW && block.timestamp > loanStartTime + loanDuration (contracts/integrations/NFTfiV2/NFTfiV2NoteAdapter.sol#241-242)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Different versions of Solidity is used:
        - 0.8.9 (contracts/integrations/NFTfiV2/NFTfiV2NoteAdapter.sol#2)
        - ^0.8.0 (contracts/interfaces/INoteAdapter.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version0.8.9 (contracts/integrations/NFTfiV2/NFTfiV2NoteAdapter.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (contracts/interfaces/INoteAdapter.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

LoanLibrary.sol

```
Pragma version^0.8.0 (contracts/integrations/ArcadeV1/LoanLibrary.sol#2) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

- No major issues were found by Slither.
- Unprotected upgradeable contract, variable shadowing, re-entrancy and divide-before-multipling issues are all false positives.

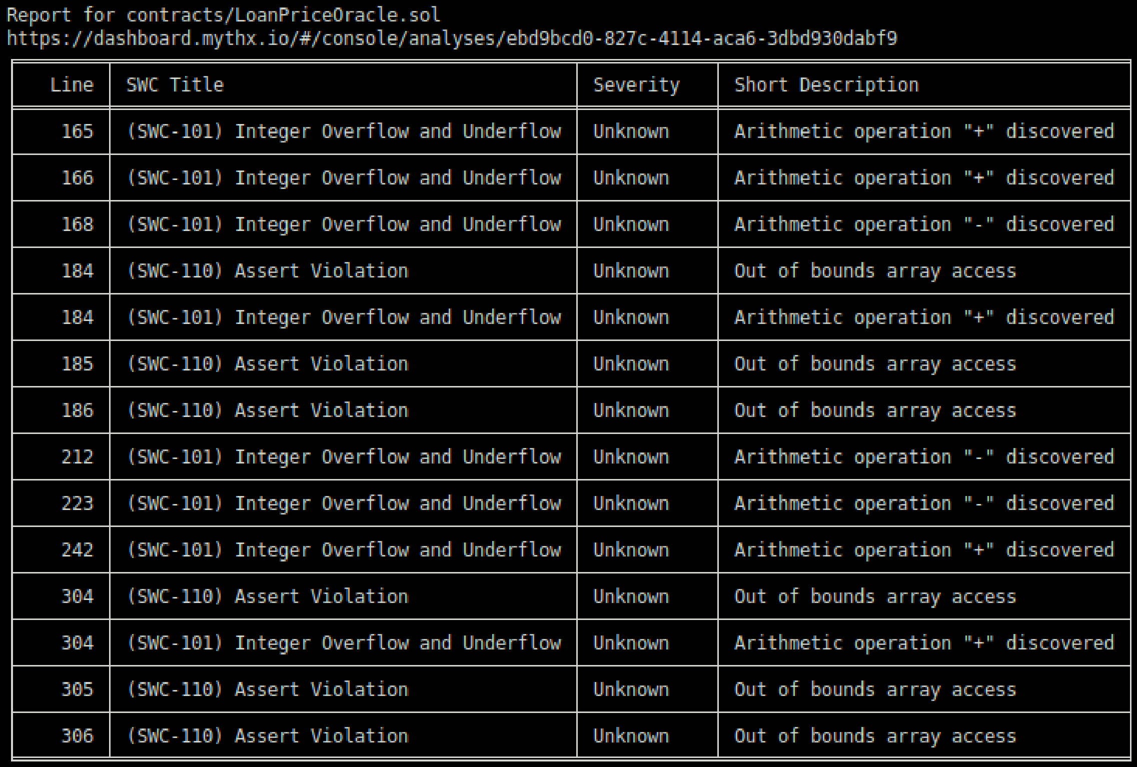
4.2 AUTOMATED SECURITY SCAN

MYTHX:

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

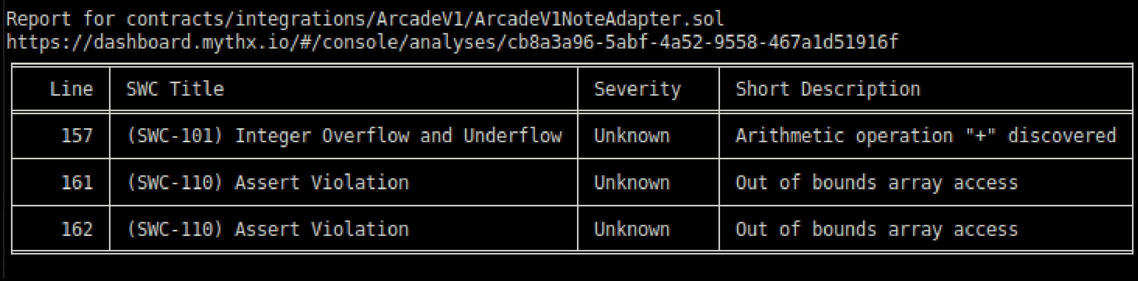
Results:

LoanPriceOracle.sol



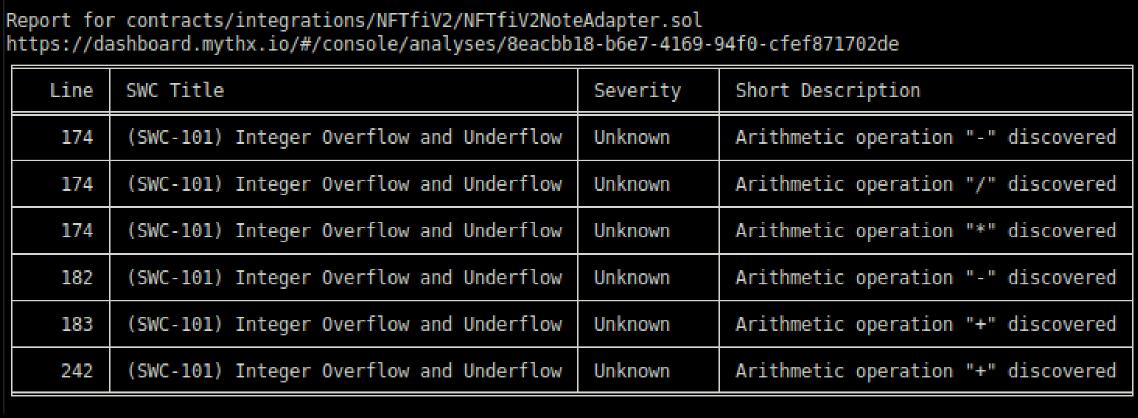
```
Report for contracts/LoanPriceOracle.sol
https://dashboard.mythx.io/#/console/analyses/ebd9bcd0-827c-4114-aca6-3dbd930dabf9
╒════════╤════════════════════════════════════════╤════════════╤════════════════════════════════════╕
│   Line │ SWC Title                              │ Severity   │ Short Description                  │
╞════════╪════════════════════════════════════════╪════════════╪════════════════════════════════════╡
│    165 │ (SWC-101) Integer Overflow and Underflow │ Unknown  │ Arithmetic operation "+" discovered │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    166 │ (SWC-101) Integer Overflow and Underflow │ Unknown  │ Arithmetic operation "+" discovered │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    168 │ (SWC-101) Integer Overflow and Underflow │ Unknown  │ Arithmetic operation "-" discovered │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    184 │ (SWC-110) Assert Violation             │ Unknown    │ Out of bounds array access         │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    184 │ (SWC-101) Integer Overflow and Underflow │ Unknown  │ Arithmetic operation "+" discovered │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    185 │ (SWC-110) Assert Violation             │ Unknown    │ Out of bounds array access         │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    186 │ (SWC-110) Assert Violation             │ Unknown    │ Out of bounds array access         │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    212 │ (SWC-101) Integer Overflow and Underflow │ Unknown  │ Arithmetic operation "-" discovered │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    223 │ (SWC-101) Integer Overflow and Underflow │ Unknown  │ Arithmetic operation "-" discovered │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    242 │ (SWC-101) Integer Overflow and Underflow │ Unknown  │ Arithmetic operation "+" discovered │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    304 │ (SWC-110) Assert Violation             │ Unknown    │ Out of bounds array access         │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    304 │ (SWC-101) Integer Overflow and Underflow │ Unknown  │ Arithmetic operation "+" discovered │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    305 │ (SWC-110) Assert Violation             │ Unknown    │ Out of bounds array access         │
├────────┼────────────────────────────────────────┼────────────┼────────────────────────────────────┤
│    306 │ (SWC-110) Assert Violation             │ Unknown    │ Out of bounds array access         │
╘════════╧════════════════════════════════════════╧════════════╧════════════════════════════════════╛
```

ArcadeV1NoteAdapter.sol



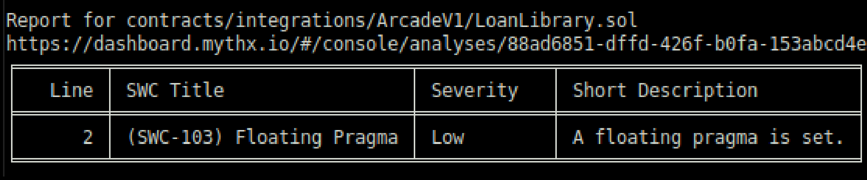
```
Report for contracts/integrations/ArcadeV1/ArcadeV1NoteAdapter.sol
https://dashboard.mythx.io/#/console/analyses/cb8a3a96-5abf-4a52-9558-467a1d51916f

| Line | SWC Title                              | Severity | Short Description                   |
|  157 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "+" discovered |
|  161 | (SWC-110) Assert Violation             | Unknown  | Out of bounds array access          |
|  162 | (SWC-110) Assert Violation             | Unknown  | Out of bounds array access          |
```

NFTfiV2NoteAdapter.sol



```
Report for contracts/integrations/NFTfiV2/NFTfiV2NoteAdapter.sol
https://dashboard.mythx.io/#/console/analyses/8eacbb18-b6e7-4169-94f0-cfef871702de

| Line | SWC Title                                | Severity | Short Description                   |
|  174 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "-" discovered |
|  174 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "/" discovered |
|  174 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "*" discovered |
|  182 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "-" discovered |
|  183 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "+" discovered |
|  242 | (SWC-101) Integer Overflow and Underflow | Unknown  | Arithmetic operation "+" discovered |
```

LoanLibrary.sol



```
Report for contracts/integrations/ArcadeV1/LoanLibrary.sol
https://dashboard.mythx.io/#/console/analyses/88ad6851-dffd-426f-b0fa-153abcd4ea11

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

- No major issues found by MythX.
- Assert violations are all false positives.
- Integer Overflows and Underflows flagged by MythX are false positives, as all the contracts are using Solidity ˆ0.8.0 version. After the Solidity version 0.8.0 Arithmetic operations revert to underflow and overflow by default.

THANK YOU FOR CHOOSING

// HALBORN