



Three Sigma Labs

Code Audit



MAPLE

Maple Finance Lending Protocol



Disclaimer

Code Audit

Maple Finance Lending Protocol

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Maple Finance Lending Protocol

Table of Contents

Disclaimer	2
Table of Contents	3
Table of Contents	4
Summary	7
Scope	9
V1 to V2 Protocol Liquidity Migration Contracts	9
V2 Protocol Contracts	9
Project Dashboard	14
Code Maturity Evaluation	17
Code Maturity Evaluation Guidelines	17
Code Maturity Evaluation Results	18
Automated Testing and Verification	20
Findings	22
3S-MAPLE-01	22
3S-MAPLE-02	24
3S-MAPLE-03	25
3S-MAPLE-04	26
3S-MAPLE-05	28
3S-MAPLE-06	30
3S-MAPLE-07	32
3S-MAPLE-08	33
3S-MAPLE-09	34
3S-MAPLE-10	35
3S-MAPLE-11	36
3S-MAPLE-12	37
3S-MAPLE-13	38
3S-MAPLE-14	39
3S-MAPLE-15	40
3S-MAPLE-16	41

	5
3S-MAPLE-17	42
3S-MAPLE-18	43

Summary

Code Audit

Maple Finance Lending Protocol

Summary

Three Sigma Labs audited Maple Finance's V2 smart contracts in a 5 person week engagement. The audit was conducted from 24-10-2022 to 09-11-2022.

Protocol Description

Maple Finance is a decentralized credit market that uses blockchain technology to allow institutional borrowers to obtain undercollateralized loans from lending pools managed by accredited delegates. This platform enables borrowers to access loans that they may not have been able to obtain elsewhere, while also providing lenders with a way to earn sustainable returns on their assets. Delegates who create and manage pools on Maple earn a portion of the establishment fee and interest earned on the loans, and liquidity providers who add funds to the pools earn a share of the interest paid by borrowers.

00000010	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	...
00000020	00 00 00 00 3B A3 ED FD	7A 7B 12 82 7A C7 2C 3E	...
00000030	67 76 8F 61 7F C8 1B C3	88 8A 51 32 3A 9F B8 AA	gv.a.Ê.Ã°SQ2 Y,
00000040	4B 1E 5E 4A 29 AB 5F 49	FF FF 00 1D 1D AC 2B 7C	K.°J)«_Iyy...~+
00000050	01 01 00 00 00 01 00 00	00 00 00 00 00 00 00 00
00000060	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000070	00 00 00 00 00 00 FF FF	FF FF 4D 04 FF FF 00 1DÿÿÿÿM.ÿÿ.
00000080	01 04 45 54 68 65 20 54	69 6D 65 73 20 30 33 2F	..EThe Times 0
00000090	4A 61 6E 2F 32 30 30 39	20 43 68 61 6E 63 65 6C	Jan/2009 Chance
000000A0	6C 6F 72 20 6F 6E 20 62	72 69 6E 6B 20 6F 66 20	lor on brink of
000000B0	73 65 63 6F 6E 64 20 62	61 69 6C 6F 75 74 20 66	second bailout f
000000C0	6F 72 20 62 61 6E 6B 73	FF FF FF FF 01 00 F2 05	or banksÿÿÿÿ..ò.
000000D0	2A 01 00 00 00 43 41 04	67 8A FD B0 FE 55 48 27	*....CA.gšÿ°pUH'
000000E0	19 67 F1 A6 71 30 B7 10	5C D6 A8 28 E0 39 09 A6	.gñ q0°.\\Ö"(àø.
000000F0	79 62 E0 EA 1F 61 DE B6	49 F6 BC 3F 4C EF 38 C4	ybâê.ab¶IÖ¼?Lî8Ä
00000100	F3 55 04 E5 1E C1 12 DE	5C 38 4D F7 BA 0B 8D 57	óU.ă.Á.þ\8M÷°...w
00000110	8A 4C 70 2B 6B F1 1D 5F	AC 00 00 00 00	šLp+kñ._~....
00000000	30 31 30 30 30 30 30 30	36 66 65 32 38 63 30 61	01000000006fe28ca
00000010	62 36 66 31 62 33 37 32	63 31 61 36 61 32 34 36	b6f1b3726c1a6a26
00000020	61 65 36 33 66 37 34 66	39 33 31 65 38 33 36 35	ae63f7f463933135
00000030	65 31 35 61 30 38 39 63	36 38 64 36 31 39 30 30	e15a089c638d6190
00000040	35 31 30 30 30 30 30 30	39 38 32 30 35 31 66 64	000000009820516fd
00000050	31 65 34 62 61 37 34 34	62 62 62 65 36 38 30 65	1e4ba7445bb6e80e
00000060	31 66 65 65 31 34 36 37	37 62 61 31 61 33 63 33	1fee1456376a13c3
00000070	85 34 31 32 66 37 62 31	63 64 62 36 30 36 65 38	5400bfb1bcd606e8
00000080	35 37 32 33 33 65 30 65	36 31 62 63 36 36 34 39	57233e0e6bc63649
00000090	66 66 66 66 30 31 64	30 31 65 33 36 32 39 39	fff0001d01e36299
000000A0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	01010000000010000
000000B0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	00000000000000000
000000C0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	00000000000000000
000000D0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	00000000000000000
000000E0	30 30 30 30 30 30 30 30	30 30 30 30 66 66 66 66	0000000000000ffff
000000F0	66 66 66 66 30 37 30 34	66 66 66 66 30 30 31 64	ffff074fff00001d
00000100	30 31 30 34 66 66 66 66	66 66 66 66 30 31 30 30	0104ffffffffff0100
00000110	66 32 30 35 32 61 30 31	30 30 30 30 30 30 34 33	f2052a01000000043
00000120	34 31 30 34 39 36 62 35	33 38 65 38 35 33 35 31	4104946b538e5853
00000130	39 63 37 32 36 61 32 63	39 31 65 36 31 65 63 31	9c726ac91e6e6ec1
00000140	31 36 30 30 61 65 31 33	39 30 38 31 33 61 36 32	16000ae139038132
00000150	37 63 36 36 66 62 38 62	65 37 39 34 37 62 65 36	7c66fb8be7947be6
00000160	33 63 35 32 64 61 37 35	38 39 33 37 39 35 31 35	3c52da7358979515
00000170	64 34 65 30 61 36 30 34	66 38 31 34 31 37 38 31	d4e0a64638141781
00000180	65 36 32 32 39 34 37 32	31 31 36 36 62 66 36 32	e622494732116662
00000190	31 65 37 33 61 38 32 63	62 66 32 33 34 32 63 38	1e73a823cbf234c8
000001A0	35 38 65 65 61 63 30 30	30 30 30 30 30 30	58eacc0000000000
00000000	30 31 30 30 30 30 30 30	34 38 36 30 65 62 31 38	0100000004860eb18
00000010	62 66 31 62 31 36 32 30	65 33 37 65 39 34 39 30	bf1b1206e3e94909
00000020	66 63 38 61 34 32 37 35	31 34 34 31 36 66 64 37	fc8a423351e44fd7
00000030	35 31 35 39 61 62 38 36	36 38 38 65 39 61 38 33	519ab8688e9a838

Scope

Code Audit

Maple Finance Lending Protocol

Scope

The audit examined Maple Finance's V2 core contracts as well as the liquidity migration contracts and procedure from V1 to V2.

V1 to V2 Protocol Liquidity Migration Contracts

maple-labs/debt-locker (v4.0.0-rc.0)

maple-labs/loan (v3.0.1-rc.0)

maple-labs/loan (v3.0.2-rc.0)

maple-labs/migration-helpers (v1.0.0-rc.1)

V2 Protocol Contracts

maple-labs/globals-v2 (v1.0.0-rc.0)

maple-labs/liquidations (v2.0.0-rc.1)

maple-labs/loan (v4.0.0-rc.1)

maple-labs/maple-proxy-factory (v1.1.0-rc.0)

maple-labs/pool-v2 (v1.0.0-rc.1)

maple-labs/withdrawal-manager (v1.0.0-rc.1)

The review was conducted on the code present in a private repository shared with Three Sigma, which contains a Foundry project with testing scripts as well as a documentation providing additional information. The code was frozen for review at commit 652d1816238e62f44cd270feb7b20ae9ca82b51f on the **maple-labs/maple-core-v2** repository which contained all the relevant modules described above.

Assumptions

The scope of the audit was carefully defined to include the contracts at the lowest level of the inheritance hierarchy, as these are the ones that will be deployed to the mainnet. No external libraries were utilized in the implementation of these contracts, so all of the relevant code was subject to review during the audit process. It is important to note that the dependencies of these contracts have been previously audited by other audit firms.



Methodology

Code Audit

Maple Finance Lending Protocol

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/.

Level	Description
Critical	<ul style="list-style-type: none">- Empty or freeze the contract's holdings.- Cryptographic flaws.
High	<ul style="list-style-type: none">- Token holders temporarily unable to transfer holdings.- Users spoof each other.- Theft of yield.- Transient consensus failures.
Medium	<ul style="list-style-type: none">- Contract consumes unbounded gas.- Block stuffing.- Griefing denial of service.- Gas griefing.
Low	<ul style="list-style-type: none">- Contract fails to deliver promised returns, but doesn't lose value.
None	<ul style="list-style-type: none">- Best practices.- Gas optimizations.



Project Dashboard

Code Audit

Maple Finance Lending Protocol

Project Dashboard

Application Summary

Name	Maple Finance
Commit	652d1816
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	24 October to 9 November, 2022
N° of Auditors	2
Review Time	5 person weeks

Vulnerability Summary

N° Critical Severity Issues	0
N° High Severity Issues	1
N° Low Severity Issues	2
N° Informational Severity Issues	Several

Category Breakdown

Suggestion	7
Optimization	7
Bug	3
Access Control	1

Code Maturity Evaluation

Code Audit

Maple Finance Lending Protocol

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Weak. Several centralized actors have significant privileges over the protocol.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Moderate. Certain smart contract implementations can be modified after deployment, albeit with proper timelocks and functional upgradeability patterns.
Function Composition	Satisfactory. Certain components are similar, and the codebase would benefit from increased code reuse.
Front-Running	Moderate. Pool delegate loan actions can be front-run, allowing users to queue up withdrawals. Front-running vulnerability when initializing shares of a pool.
Monitoring	Satisfactory. Events are correctly emitted.
Specification	Satisfactory. In-depth and well structured high-level specification as well as codebase documentation.
Testing and Verification	Satisfactory. Extensive test code coverage as well as usage of tools and different test methods.

Automated Testing and Verification

Code Audit

Maple Finance Lending Protocol

Automated Testing and Verification

To enhance coverage of certain areas of the codebase we complement our analysis with a set of automated testing techniques:

- **Slither:** A Solidity static analysis framework with native support for multiple vulnerability detectors. We used Slither to scan the entire codebase against common vulnerabilities and programming malpractices.

Despite augmenting our security analysis, automated testing techniques still present some limitations and should therefore not be used in isolation. Slither may fail to identify vulnerabilities, either due to the lack of specific detectors or whenever certain properties fail to hold after Solidity code is compiled to EVM bytecode.

Slither Results

An initial run of Slither on Maple Finance's V2 smart contracts reported warnings considered to be false positives. Maple's codebase was equipped with automated scripting that ran Slither and the team already incorporated Slither's outputs into the development phases of the code.

00000010	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00000020	00 00 00 00 30 A3 ED FD	7A 7B 12 82 7A C7 2C 3E
00000030	67 76 8F 61 7F C8 1B C3	88 8A 51 32 3A 9F B8 AA	00000040	4B 1E 5E 4A 29 AB 5F 49	FF FF 00 1D 1D AC 2B 7C
00000050	01 01 00 00 00 01 00 00	00 00 00 00 00 00 00 00	00000060	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000070	00 00 00 00 00 00 FF FF	FF FF 4D 04 FF FF 00 1D	00000080	01 04 45 54 68 65 20 54	69 6D 65 73 20 30 33 2F
00000090	4A 61 6E 2F 32 30 30 39	20 43 68 61 6E 63 65 6C	000000A0	6C 6F 72 20 6F 6E 20 62	72 69 6E 6B 20 6F 66 20
000000B0	73 65 63 6F 6E 64 20 62	61 69 6C 6F 75 74 20 66	000000C0	6F 72 20 62 61 6E 6B 73	FF FF FF FF 01 00 F2 05
000000D0	2A 01 00 00 00 43 41 04	67 8A FD B0 FE 55 48 27	000000E0	19 67 F1 A6 71 30 B7 10	5C D6 A8 28 E0 39 09 A6
000000F0	79 62 E0 EA 1F 61 DE B6	49 F6 BC 3F 4C EF 38 C4	00000100	F3 55 04 E5 1E C1 12 DE	5C 38 4D F7 BA 0B 8D 57
00000110	8A 4C 70 2B 6B F1 1D 5F	AC 00 00 00 00	00000000	30 31 30 30 30 30 30 30	36 66 65 32 38 63 30 61	01000000006fe28ca	
00000010	62 36 66 31 62 33 37 32	63 31 61 36 61 32 34 36	00000020	61 65 36 33 66 37 34 66	39 33 31 65 38 33 36 35	ae63f7f463933135	
00000030	65 31 61 30 38 39 63	36 38 64 36 31 39 30 30	00000040	00 00 00 00 00 00 00 30	39 38 32 30 35 31 66 64	e15a089c638d6190	
00000050	31 65 34 62 67 34 34	62 62 62 65 36 38 30 65	00000060	31 66 65 65 31 34 36 37	37 62 61 31 61 33 63 33	1e4ba7445bb6e80e	
00000070	85 34 32 66 37 62 31	63 64 62 36 30 36 65 38	00000080	33 37 32 33 33 65 30 65	36 31 62 63 36 36 34 39	1fee1456376a13c3	
00000090	66 66 66 66 30 30 31 64	30 31 65 33 36 32 39 39	000000A0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	5400bfb1bcd606e8	
000000B0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000C0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	57233e0e6bc63649	
000000D0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000E0	30 30 30 30 30 30 30 30	30 30 30 30 66 66 66 66	fff0001d01e36299	
000000F0	66 66 66 66 30 37 30 34	66 66 66 66 30 30 31 64	00000100	30 31 30 34 66 66 66 66	66 66 66 66 30 31 30 30	0104fffffffff0100	
00000110	66 32 30 35 32 61 30 31	30 30 30 30 30 30 34 33	00000120	34 31 30 34 39 36 62 35	33 38 65 38 35 33 35 31	f2052a0100000043	
00000130	39 63 37 32 36 61 32 63	39 31 65 36 31 65 63 31	00000140	31 36 30 30 61 65 31 33	39 30 38 31 33 61 36 32	4104946b538e5853	
00000150	37 63 36 36 66 62 38 62	65 37 39 34 37 62 65 36	00000160	33 63 35 32 64 61 37 35	38 39 33 37 39 35 31 35	9c726ac91e6e6ec1	
00000170	64 34 65 30 61 36 30 34	66 38 31 34 31 37 38 31	00000180	65 36 32 32 39 34 37 32	31 31 36 36 62 66 36 32	16000ae139038132	
00000190	31 65 37 33 61 38 32 63	62 66 32 33 34 32 63 38	000001A0	35 38 65 65 61 63 30 30	30 30 30 30 30 30	7c66fb8be7947be6	
00000000	30 31 30 30 30 30 30 30	34 38 36 30 65 62 31 38	00000010	62 66 31 62 31 36 32 30	65 33 37 65 39 34 39 30	3c52da7358979515	
00000020	66 63 38 61 34 32 37 35	31 34 34 31 36 66 64 37	00000030	35 31 35 39 61 62 38 36	36 38 38 65 39 61 38 33	d4e0a64638141781	
00000040			00000050			58eacc0000000000	
00000060			00000070			010000004860eb18	
00000080			00000090			bf1b1206e3e94909	
000000A0			000000B0			fc8a423351e44fd7	
000000C0			000000D0			519ab8688e9a838	

Findings

Code Audit

Maple Finance Lending Protocol

Findings

3S-MAPLE-01

Loss of funds due to poor initialization of pool shares accounting

Id	3S-MAPLE-01
Severity	High
Difficulty	Medium
Category	Bug

Description

When depositing into the `Pool` contract, deposited assets are converted into vault shares according to the following formula, where `totalSupply_` corresponds to the number of existing shares:

```
shares_ = totalSupply_ == 0 ? assets_ : (assets_ * totalSupply_) / totalAssets();
```

In the first case, when no shares are circulating, the number of minted vault shares is equal to the amount of assets deposited. In the second case the depositor gets minted an amount of shares proportional to its percentage of total liquidity (rounded down).

This allows for a scenario where an attacker might steal a user deposit by frontrunning the initial deposit into a vault and subsequently inflating the vault's assets.

The `Pool` contract has a guard that requires the number of minted shares to be larger than

zero, otherwise reverting with a `P:M:ZERO_SHARES` message. This check can be bypassed by adjusting the variables under the attacker's control (`totalSupply_` and `totalAssets()`) in order to force the rounding down for a number other than zero.

Recommendation

After deploying a new pool, require the first mint to mint a minimum number of shares that mitigates this possible attack vector.

Status

Addressed in the following PRs:

<https://github.com/maple-labs/pool-v2/pull/219>

<https://github.com/maple-labs/globals-v2/pull/40>

3S-MAPLE-02

Inconsistent values on `LoanImpaired` events

Id	3S-MAPLE-02
Severity	Low
Difficulty	Low
Category	Bug

Description

During the loan impairment flow, initiated by a call to the `impairLoan` function of the `PoolManager` contract, a `LoanImpaired` event is emitted by both the `PoolManager.sol` and the `MapleLoan.sol` contracts. Despite having distinct signatures, both events share a `nextPaymentDueDate_` field, which is set differently on each emission.

In the `PoolManager.sol` contract, the value is always set as the `block.timestamp`, while in a `MapleLoan` contract, it is calculated according to the following formula:

```
newPaymentDueDate_ = block.timestamp > originalNextPaymentDueDate_ ?  
originalNextPaymentDueDate_ : block.timestamp
```

Recommendation

Update the variables on the events being emitted to be coherent between different uses.

Status

Addressed by the team.

3S-MAPLE-03

The Pool Manager temporarily loses the ability to transfer from the `PoolDelegate` contract

Id	3S-MAPLE-03
Severity	Low
Difficulty	High
Category	Access Controls

Description

If Pool Delegates of two `PoolManager.sol` contracts set the same address for the variable `pendingPoolDelegate` and then the new Pool Delegate calls the function `acceptPendingPoolDelegate` on both PMs, then the first accepted `PoolManager` becomes unable to set a new Pool Delegate. It is possible to recover this functionality if the new Pool Delegate sets himself as the `pendingPoolDelegate` on the first `PoolManager` contract and then accepts it, but in doing so the functionality on the second PM is lost. Summarizing, the map variable `poolDelegates` on `GlobalsV2` contract only allows a Pool Delegate to change Pool Delegates for one `PoolManager` at a time.

Recommendation

Update the data structure to allow a single Pool Delegate to be the Pool Delegate of multiple `PoolManager` contracts.

Status

Acknowledged by the team.

3S-MAPLE-04

Use linked list insertion hints

Id	3S-MAPLE-04
Severity	None
Difficulty	N/A
Category	Optimization

Description

The `LoanManager.sol` contract uses linked lists to store stored loan payment information according to their due dates. In order to insert a new payment in the linked list, the `_addPaymentToList(...)` function iterates over all the elements in the list. It begins iterating from the start until the due date of the payment being inserted is smaller than that of the list element. This operation has a complexity of $O(N)$, where “N” is the number of elements in the list.

The same behavior occurs in the `PoolManager` contract. The function `removeLoanManager(...)` iterates through the elements of the `loanManagerList` until it finds the correct element to be removed.

A way to improve the complexity of the search is through the use of insertion hints. Insertion hints are computed off chain and passed as calldata to the Maple protocol contracts. As an example of an insertion hint, we can consider the variables `prevId_` and `nextId_`, which give information regarding the expected previous and next nodes of the inserted payment, upon insertion. Instead of iterating over the list to find a valid insert position, the `_addPaymentToList(...)` or the function `removeLoanManager(...)` should simply verify that the position specified by the insertion hints is valid, meaning:

- The payment at `_prevId` has a due time smaller than the one being inserted.

- The payment at `_nextId` has a due time larger than the one being inserted.
- The payment at `_prevId` points to the payment at `_nextId` (before insertion).

Note that this is just a simple example of an algorithm. You can also build a more efficient one using a single hint value.

Nonetheless, the algorithm has an $O(1)$ complexity, which is far superior to the original $O(N)$ complexity.

Recommendation

Modify the `_addPaymentToList(...)` and `removeLoanManager(...)` function signatures to accept insertion hints.

Status

Acknowledged by the team.

3S-MAPLE-05

Disruption of withdrawals due to pro rata distribution of shares

Id	3S-MAPLE-05
Severity	None
Difficulty	N/A
Category	Suggestion

Description

Users that request a withdrawal during cycle n will be able to withdraw during the withdrawal window at the beginning of cycle $n+2$ since they must wait at least one full cycle from the end of the cycle they locked their shares. All withdrawals grouped in a specific cycle may request a pro rata distributed portion of the pool's existing funds to withdraw.

This opens an attack vector where a malicious user can queue a very large amount of capital to request a withdrawal without any intention of withdrawing any amount in that cycle, diluting the other withdrawal requests by everyone else. This can then be done for every cycle, making it impossible for anyone to get their money back. To mitigate this, the Maple team makes it a pre-requirement for a withdrawal request to be carried across cycles, for the user to attempt to finalize the withdrawal for its full amount, and for it to have been partially filled if there isn't enough liquidity in the pool. If the user doesn't try to withdraw their assets, the request can only be fulfilled in round $n+2$, allowing other users to withdraw their funds in round $n + 1$.

However, under the assumption that the adversary has "unlimited" capital, it would be possible to make a large withdrawal targeting even cycles and one withdrawal targeting odd cycles, in turn covering all rounds as queued requests can always be carried to round $n+2$.

Recommendation

Although the likelihood of this situation occurring is very low, it should be taken into consideration when considering potential improvements to the mechanism design that is currently implemented.

Status

Acknowledged by the team.

3S-MAPLE-06

Pool Delegate is able to block withdrawals

Id	3S-MAPLE-06
Severity	None
Difficulty	N/A
Category	Suggestion

Description

In the `WithdrawalManager.sol` contract in the `lockedLiquidity` function from contract calculates the current locked liquidity to be used in withdrawals, which cannot be used to fund open loans. It returns the distribution of available assets with the amount of shares locked for the current withdrawal window.

Users that request a withdrawal during cycle n will be able to withdraw during the withdrawal window at the beginning of cycle $n+2$ since they must wait at least one full cycle from the end of the cycle in which they locked their shares. Users are only able to withdraw during a withdrawal window, which starts at the beginning of each cycle.

This means that users must commit funds for up to almost three full cycles, while the pool delegate is only restricted from using these assets to fund loans during the last cycle's window, when the withdrawals mature. This can result in a scenario where the pool delegate can indefinitely lock liquidity into the protocol by funding loans with funds queued for withdrawal.

Recommendation

Although the likelihood of this situation occurring is very low, it should be taken into consideration when considering potential improvements to the mechanism design that is currently implemented.

Status

Acknowledged by the team.

3S-MAPLE-07

The `require` check is out of order

Id	3S-MAPLE-07
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `Liquidator.sol` contract in the function `liquidatePortion` the statements

```
uint256 returnAmount = getExpectedAmount(collateralAmount_);  
require(returnAmount <= maxReturnAmount_, "LIQ:LP:MAX_RETURN_EXCEEDED");
```

are not at the beginning of the function, which incurs unnecessary gas costs when the check inside the `require` is false.

Recommendation

Move these statements to the beginning of the function.

Status

Acknowledged by the team, will fix at a later date.

3S-MAPLE-08

Different modifiers for same functionality

Id	3S-MAPLE-08
Severity	None
Difficulty	N/A
Category	Suggestion

Description

The `Liquidator.sol` contract uses a `Lock` modifier while the `LoanManager.sol` contract uses a `nonReentrant` modifier to prevent reentrancy on particular functions. These implementations of the same functionality are redundant and not standardized.

Recommendation

Use a single modifier to prevent reentrant behavior across the codebase.

Status

Addressed in the following PR:

<https://github.com/maple-labs/3sigma-audit-2022-10-24/issues/17>

3S-MAPLE-09

Function rename suggestion

Id	3S-MAPLE-09
Severity	None
Difficulty	N/A
Category	Suggestion

Description

The function `_isCollateralMaintained` in `MapleLoan.sol` contract could be called `_isCollateralAmountMaintained`. The original function name may be confusing because such naming is commonly seen on codebases to indicate whether a specific collateral is supported.

Recommendation

Rename function to `_isCollateralAmountMaintained`.

Status

Acknowledged by the team.

3S-MAPLE-10

Variable naming convention inconsistency

Id	3S-MAPLE-10
Severity	None
Difficulty	N/A
Category	Suggestion

Description

The `_` suffix is used throughout the codebase whenever a variable is instantiated to memory. In the `MapleLoan.sol` contract, in the function `repossess`, the variable `nextPaymentDueDateCache`'s name is inconsistent since it uses `Cache` as a suffix instead.

Recommendation

Change to the `_` suffix.

Status

Acknowledged by the team, will be addressed at a later date.

3S-MAPLE-11

Redundant variable assignment

Id	3S-MAPLE-11
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `LoanManager.sol` contract in the function `triggerDefault` the assignment `liquidationComplete_ = false;` is redundant and can be safely removed.

Recommendation

Remove assignment.

Status

Addressed in the following PR:

<https://github.com/maple-labs/pool-v2/pull/193>

3S-MAPLE-12

The `require` check is out of order

Id	3S-MAPLE-12
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the function `_disburseLiquidationFunds` in the `LoanManager.sol` contract the statements

```
address mapleTreasury_ = mapleTreasury();  
require(mapleTreasury_ != address(0), "LM:DLF:ZERO_ADDRESS");
```

are not at the beginning of the function, which incurs in unnecessary gas costs when the check inside the `require` is false.

Recommendation

Move these statements to be the first ones in the function.

Status

Acknowledged by the team, will address at a later date.

3S-MAPLE-13

Function `_revertLoanImpairment` doesn't emit event

Id	3S-MAPLE-13
Severity	None
Difficulty	N/A
Category	Bug

Description

The function `_revertLoanImpairment` in the `LoanManager.sol` contract does not emit an update after changing the `unrealizedLosses` variable. This function is called in 2 places:

1. `removeLoanImpairment` in which the event `UnrealizedLossesUpdated` is emitted after the function call;
2. `_handlePreviousPaymentAccounting` where the event is not emitted.

Recommendation

Emit the event inside the internal function.

Status

Acknowledged by the team.

3S-MAPLE-14

Access to variable in storage instead of memory

Id	3S-MAPLE-14
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `MapleLoan.sol` contract in the `makePayment` function the following statement:

```
ILenderLike(_lender).claim(principal_, interest_, previousPaymentDueDate_,  
_nextPaymentDueDate);
```

uses `_nextPaymentDueDate` instead of `nextPaymentDueDate_` which incurs extra gas costs.

Recommendation

Change the variable used.

Status

Addressed in the following PR:

<https://github.com/maple-labs/loan/pull/237>

3S-MAPLE-15

Unnecessary variable and `require`

Id	3S-MAPLE-15
Severity	None
Difficulty	N/A
Category	Suggestion

Description

In the `WithdrawalManager` contract the `processExit` function receives a `requestedShares_` amount passed as argument. This argument is required to match exactly the amount of shares locked by the account requesting the withdrawal for the current withdrawal window. This parameter is not used for anything else so it is redundant and it should be removed.

Recommendation

Remove the `requestedShares_` argument and replace it with the `lockedShares_` value in `processExit()`.

Status

Acknowledged by the team.

3S-MAPLE-16

Replace reason strings with custom errors

Id	3S-MAPLE-16
Severity	None
Difficulty	N/A
Category	Optimization

Description

To provide more information about failures, the Maple protocol implementation employs reason strings (for example, `revert("LM:RLI:NOT_AUTHORIZED")`). These are considered to be:

1. rather expensive, especially when it comes to deployment cost,
2. difficult to use with dynamic information
3. and incur extra costs when the revert condition is met.

Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors.

Recommendation

Consider replacing all instances of the reason string require statements with custom errors across the codebase.

Status

Acknowledged by the team.

3S-MAPLE-17

Missing migration admin value check

Id	3S-MAPLE-17
Severity	None
Difficulty	N/A
Category	Suggestion

Description

In the `_initialize(...)` function of `PoolManagerInitializer.sol()` contract the recipient of the new LP tokens, minted upon pool V2 deployment, is set to be the migration admin, read from storage `IMapleGlobalsLike(globals_).migrationAdmin()`.

The pool manager initialization phase of the Maple Finance V2 migration process is the responsibility of each pool delegate. Since the `_initialize(...)` function does not enforce that a migration admin must be configured priorly making it possible for a pool delegate to mint LP tokens to the `0x0` address, during poorly coordinated migration.

Recommendation

Add a `require` statement to ensure the migration manager is configured prior to pool manager initialization.

Status

Acknowledged by the team, will address at a later date.

3S-MAPLE-18

Redundant guard condition

Id	3S-MAPLE-18
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `TransitionLoanManager.sol` contract the first condition in the following control statement of the `add(address loan_)` function is redundant and can safely be removed. This is due to the fact that `block.timestamp` should never be equal to zero.

Recommendation

Remove the extra guard.

Status

Addressed in the following PR:

<https://github.com/maple-labs/pool-v2/pull/197>