Three Sigma

Code Audit

Maple Lending Protocol

Disclaimer

Code Audit

Maple 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 Lending Protocol

Table of Contents

Summary

Code Audit

Maple Lending Protocol

Summary

Three Sigma audited Maple in a 4 person week engagement. The audit was conducted from 25-11-24 to 6-12-24.

Protocol Description

Maple Finance is an institutional crypto-capital network, which provides the infrastructure for credit experts to efficiently manage and scale crypto lending businesses and connect capital from institutional and individual lenders to innovative, blue-chip companies.

Scope

Code Audit

Maple Lending Protocol

Scope

Filepath	Change Description
GlobalsV2/MapleGlobals.sol	Updated logic for canDeployFrom() to support strategies and compiler update
PoolV2/MaplePoolManager.sol	Updated logic to support strategies
PoolV2/MaplePoolDeployer.sol	New logic to deploy pools with strategies
PoolV2/MaplePool.sol	Compiler update only
PoolV2/MaplePoolDelegateCover.sol	Compiler update only
PoolV2/MaplePoolManagerInitializer.sol	Compiler update only
PoolV2/MaplePoolManagerStorage.sol	Updated variable names to support strategies
strategies/MapleAbstractStrategy.sol	New logic
strategies/MapleAaveStrategy.sol	New logic
strategies/MapleSkyStrategy.sol	New logic
strategies/MapleBasicStrategy.sol	New logic
strategies/MapleStrategyFactory.sol	Redeployment of previously audited factory
strategies/proxy/MapleAbstractStrategy.sol	New logic
strategies/proxy/aaveStrategy/MapleAaveStrategyInitializer.sol	New logic
strategies/proxy/aaveStrategy/MapleAaveStrategyStorage.sol	New logic
strategies/proxy/skyStrategy/MapleSkyStrategyInitializer.sol	New logic
strategies/proxy/skyStrategy/MapleSkyStrategyStorage.sol	New logic

strategies/proxy/basicStrategy/MapleBasicStrategyInitializer.sol	New logic
strategies/proxy/basicStrategy/MapleBasicStrategyStorage.sol	New logic

Globals v2: Diff since deployment: Compare v3.0.0...v3.0.1

Pool v2: Diff since deployment: Compare v3.0.0...v4.0.0

Maple Strategies: Release: v1.0.0-rc.1

Methodology

Code Audit

Maple 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/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Maple Lending Protocol

Project Dashboard

Application Summary

Name	Maple
Commit	5c3663cb950103c4db576ac2e1d962ba9c2ed340
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	25-11-24 to 6-12-24
№ of Auditors	2
Review Time	4 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	2	0	2

None	13	6	7

Category Breakdown

Suggestion	11
Documentation	0
Bug	1
Optimization	1
Good Code Practices	2

Code Maturity Evaluation

Code Audit

Maple 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. Certain roles have complete control of the funds.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Satisfactory. All contracts are upgradeable.
Function Composition	Satisfactory. Certain components are similar, and the codebase would benefit from increased code reuse.
Front-Running	Satisfactory. Some Front-Running issues were found, but they can be operationally managed.
Monitoring	Satisfactory. Events are correctly emitted and Maple Finance keeps track of on chain activity.
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.

Findings

Code Audit

Maple Lending Protocol

Findings

3S-Maple-L01

`MapleSkyStrategy:: _gemForUsds()` suffers a rounding error up to approximately **`1e12`** Usds

Id	3S-Maple-L01
Classification	Low
Severity	Low
Likelihood	High
Category	Suggestion
Status	Acknowledged

Description

`MapleSkyStrategy:: _gemForUsds()` computes **`gemAmount_ = (usdsAmount_ * WAD) / (to18ConversionFactor * (WAD + tout));`**. As can be seen, **`to18ConversionFactor`** is **`1e12`** for a Usdc gem, which leads to a rounding up to **`1e12`** Usds.

Recommendation

As the rounding error corresponds to only **`1e-6`** USD and the operation is not called frequently, it's safe to ignore it.

3S-Maple-L02

Aave and SavingsUsds strategies may revert when trying to withdraw all funds

Id	3S-Maple-L02
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

Withdrawing all the Aave or Usds balance from the strategy may revert when there is yield. It withdraws first the fee to the treasury, which internally rounds up in Aave and SavingUsds, then it withdraws the remaining of the funds, but as it rounded up when withdrawing before, it will revert.

For confirmation, run **`testFork_aaveStrategy_withdraw_withFeesAndYield()`**, change **`warp`** to **`vm.warp(block.timestamp + 1980526);`** and just before the **`withdrawFromStrategy()`** call, add **`amountToWithdraw = aaveToken.balanceOf(address(aaveStrategy)) - fee;`**.

It will revert with **`32`**, which is the error for not enough balance.

Recommendation

Either accept the issue and send a slightly smaller amount to withdraw or cap the withdrawal to the maximum withdrawal value after the fee has been collected to the treasury.

3S-Maple-N01

`MapleSkyStrategy` does not always cache the **`psm`** and misses underscores

Id	3S-Maple-N01
Classification	None
Category	Optimization
Status	Addressed in #091b09f

Description

`MapleSkyStrategy::_gemForUsds()` and **`MapleSkyStrategy::_usdsForGem()`** do not cache the **`psm`** nor add trailing underscores to **`tout`** and **`to18ConversionFactor`**.

Recommendation

Implement the fixes to ensure gas savings and correct formats.

3S-Maple-N02

Differences in the strategy implementations that could be fixed

Id	3S-Maple-N02
Classification	None
Category	Good Code Practices
Status	Addressed in #4731c88

****Description****

The **`MapleAaveStrategyStorage`** contract is missing the **`State Variables`** separator comment that is present in other similar contracts like **`MapleSkyStrategyStorage`** and **`MapleBasicStrategyStorage`**.

****Recommendation****

To maintain consistency and improve code readability across all strategy storage contracts, it's recommended to add the **`State Variables`** separator comment to the **`MapleAaveStrategyStorage`** contract.

3S-Maple-N03

`ERC4626::previewRedeem()` may revert, which will DoS `MaplePool` withdrawals

Id	3S-Maple-N03
Classification	None
Category	Suggestion
Status	Acknowledged

Description

The `EIP4626` interface says for `previewRedeem()` that it may revert, which will DoS withdrawals in the `MaplePool`:

> MAY revert due to other conditions that would also cause redeem to revert.

Recommendation

This should be analyzed on a case by case basis for each underlying `ERC4626` vault.

3S-Maple-N04

The basic strategy does not have slippage control when withdrawing which may lead to arbitrage

Id	3S-Maple-N04
Classification	None
Category	Suggestion
Status	Acknowledged

Description

`MapleBasicStrategy::withdrawFromStrategy()` does not send an upper limit on the amount of shares minted, which means an unexpected loss may happen in case of slashing before the withdrawal from the **`EIR4626`** vault. Although many vaults do not have slashing abilities (such as the one tested, SavingsUsds), some may do which can cause this issue.

Recommendation

Consider adding slippage control on a case by case basis.

3S-Maple-N05

`MapleSkyStrategy::assetsUnderManagement()` uses `maxWithdraw()`, which may return 0

Id	3S-Maple-N05
Classification	None
Category	Suggestion
Status	Addressed in #6b9ece7.

Description

`MapleSkyStrategy::assetsUnderManagement()` uses `maxWithdraw()`, which according to EIP4626 returns 0 when the withdrawal limit is reached or similar. The `SavingsUsds` contract is upgradeable, so even if it currently returns the correct value, it may be upgraded in the future, adding a withdrawal limit. This would bring concerns in regards to silently reporting 0 assets.

Recommendation

Although not a problem right now, consider calling `balanceOf()` followed by `previewRedeem()`.

3S-Maple-N06

`MapleSkyStrategy:: setPsm()` should set the old Psm's approval to 0

Id	3S-Maple-N06
Classification	None
Category	Suggestion
Status	Addressed in #ef20b1b

Description

`MapleSkyStrategy::setPsm()` changes Psm, but does not set the old Psm approval to 0.

Recommendation

Set the old Psm approval to 0 as it is no longer needed.

3S-Maple-N07

`tin` in the Psm will cause an instant drop in the share price which could be leveraged by Maple Pool users

Id	3S-Maple-N07
Classification	None
Category	Suggestion
Status	Acknowledged

Description

Currently **`tin`** is null in the **`Psm`**, but if it was non null, it would instantly lead to a reduction in the **`totalAssets()`** in the Maple Pool, as a fee would be taken when converting **`fundsAssets`** to Usds in the Sky Strategy, when requesting funds.

This would allow Maple Pool users to frontrun the **`MapleSkyStrategy::fundStrategy()`** and withdraw, avoiding the fee, making unaware users pay for this. Then, they could redeposit to keep earning fees.

The damage is limited by the withdrawal mechanism.

Recommendation

Consider procedures to safely request funds without allowing users to escape the fee.

3S-Maple-N08

There may not be enough **`gem`**(Usdc) in the Psm contract, DoSing withdrawals in the Sky Strategy

Id	3S-Maple-N08
Classification	None
Category	Suggestion
Status	Acknowledged

Description

The psm contract may not have enough **`gem`** (USDC) in the pocket, which could revert and DoS withdrawals.

Recommendation

It's possible to change the **`psm`** address so the issue can be managed, but it could DoS withdrawals for some time.

3S-Maple-N09

Usdc to Usds calculation in the Sky Strategy is slightly different than the Psm Usds Wrapper

Id	3S-Maple-N09
Classification	None
Category	Good Code Practices
Status	Addressed in #d12540a

Description

In the psmWrapper, the Usds amount from the **gem** is calculated as **`usdsInWad = gemAmt18 + gemAmt18 * psm.tout() / WAD;`** , but in the Sky Strategy it is calculated as **`(gemAmount_ * to18ConversionFactor * (WAD + tout)) / WAD;`** .

Recommendation

Although the result is the same (including the rounding error), consider implementing the same exact formula.

3S-Maple-N10

The **`DaiJoin`** contract may be caged which will DoS withdrawals forever

Id	3S-Maple-N10
Classification	None
Category	Suggestion
Status	Acknowledged

Description

The daiJoin contract may be caged, which means **`exit()`** is DoSed forever. This is called in the psmWrapper **`buyGem()`**, **`legacyDaiJoin.exit(address(this), usdsInWad);`**, so it would be forever DoSed as there is no way to 'uncage'.

Recommendation

It's possible to set a new **`Psm`** in the Sky strategy but it is something to keep in mind.

3S-Maple-N11

Withdrawals in the Maple Pool and Sky Strategy may be DoSed in case the **`DssLitePsm`** halts buying

Id	3S-Maple-N11
Classification	None
Category	Suggestion
Status	Acknowledged

Description

The DssLitePsm, used as part of the Usds Psm wrapper that allows converting **`fundsAsset`** to Usds in the Sky Strategy, may be halted by setting **`tout`** to **`type(uint256).max`**.

In this case, **`MapleSkyStrategy::assetsUnderManagement()`** would revert due to overflow when calculating the underlying balance of the strategy. As such, withdrawals would be DoSed in the Maple Pool.

Recommendation

The issue can be resolved by setting the pool to inactive, although other solutions are possible. In this case, halting withdrawals may actually be the desired outcome, but it's something to keep in mind.

3S-Maple-N12

Aave RewardsController can add Usdc to the rewards list and the strategy has no way to collect the rewards

Id	3S-Maple-N12
Classification	None
Category	Suggestion
Status	Addressed in #24d2313

Description

The Aave strategy does not deal with the reward controller, so it could miss out on rewards emitted in case the **`fundsAsset`** is listed in the RewardsController.

Recommendation

Although the RewardsController does not currently emit rewards for Usdc, which is the asset that is going to be supported, consider adding the functionality to collect rewards.

3S-Maple-N13

Inactive or Impaired pool creates arbitrage oportunities

Id	3S-Maple-N13
Classification	None
Category	Suggestion
Status	Acknowledged

Description

When **`assetsUnderManagement()`** decreases, a window of opportunity is created for people to stake in the maple pool very cheaply and then sell for profit when it becomes active again or the protocol admin calls **`withdrawFromStrategy()`**.

For inactive:

This test shows that users may stake in the maple pool and get a lot of shares, protocol admin withdraws from strategy, instantly increasing the total assets of the pool. **`pool.totalAssets()`** goes from **`poolLiquidity - amountToFund`** to **`poolLiquidity - amountToFund + amountToWithdraw`** instantly, so a user stakes like 1000, gets 1000 shares, protocol admin withdraws from strategy and then user redeems for 1500 assets.

For impairment:

This test shows that users can do the same thing, but the profit is much smaller, as only the fee in **`pool.totalAssets()`** differs from before and after **`withdrawFromStrategy()`**.

Recommendation

This can be managed operationally by disabling deposits in the pool while the strategy is impaired/inactive.