Three Sigma

Code Audit

MORE

MORE Morpho Markets Automated Optimized Loops

Disclaimer

Code Audit

MORE Morpho Markets Automated Optimized Loops

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

MORE Morpho Markets Automated Optimized Loops

Table of Contents

Summary

Code Audit

MORE Morpho Markets Automated Optimized Loops

Summary

Three Sigma audited MORE in a 3 person days engagement. The audit was conducted from 02–12-2024 to 06-12-2024.

Protocol Description

MORE Optimizer v1 is designed to consume idle liquidity on correlated asset markets on Morpho such as wETH-wstETH in order to execute loops on the collateral asset. The protocol provides amplified yield for passive depositors, while optimizing underlying markets for target utilization, ensuring competitive rates for lenders. The protocol can automatically adjust the number of loops based on market utilization.

Scope

Code Audit

MORE Morpho Markets Automated Optimized Loops

Scope

Filepath	nSLOC
LoopStrategy.sol	676
ProtocolFeeManager.sol	39

Assumptions

OpenZeppelin is considered secure.

Methodology

Code Audit

MORE Morpho Markets Automated Optimized Loops

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

MORE Morpho Markets Automated Optimized Loops

Project Dashboard

Application Summary

Name	MORE
Commit	37107cf335b637e1a4af8977a671bed7155dbb18
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	02-12-2024 to 06-12-2024
№ of Auditors	1
Review Time	3 person days

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	2	2	0
High	1	1	0
Medium	2	1	1
Low	2	1	1

None	6	6	0

Category Breakdown

Suggestion	3
Documentation	0
Bug	8
Optimization	0
Good Code Practices	2

Code Maturity Evaluation

Code Audit

MORE Morpho Markets Automated Optimized Loops

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. No arithmetic errors were found.
Centralization	Moderate. The owner has control over some functionality, but most of the design is decentralized.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Satisfactory. The contracts are upgradeable.
Function Composition	Satisfactory. Functionality is well split into helper functions.
Front-Running	Satisfactory. No front-running issues were found.
Monitoring	Satisfactory. Events were 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.

Findings

Code Audit

MORE Morpho Markets Automated Optimized Loops

Findings

3S-MORE-C01

Funds may be stolen by calling **`onMoreFlashLoan()`** directly

Id	3S-MORE-C01
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #7d71809.

Description

`LoopStrategy::onMoreFlashLoan()` does not validate that the caller is the market, which allows attackers to steal collateral by calling it directly.

Recommendation

Revert if the caller is not **`markets`**.

3S-MORE-C02

Malicious **`path`** can be passed to **`redeem/withdraw()`** allowing an attacker to draing the strategy

Id	3S-MORE-C02
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #ef31c48, #d9a89a9.

Description

`path` is not validated in **`redeem/withdraw()`**, which could lead to problems as the flow should always sell and buy the same token.

Note: if a reentrant token is passed in the path it may be possible to deposit in the **`LoopStrategy`** while swapping, which would lead to stolen funds as the ratio would be higher due to having burned the shares but the assets are still in the strategy.

Note2: if **`ankrFlow`** is reentrant it could also be used to drain the funds.

For example, consider this path:

- attacker creates a pool of wFlow/reentrantToken and ankrFlow/reentrantToken;

- attacker deposits large amount of funds in the **`LoopStrategy`**;

- attacker withdraws using the following path:

  - ankrFlow → reentrantToken

  - reentrantToken → wFlow;

- whilst swapping, the reentrantToken allows the attacker to reenter the strategy and deposit. This deposit will give inflated shares as the **`_withdraw()`** method burns the shares first, but only withdraws all the assets by the end (the vault, for example);

- the attacker withdraws the inflated shares for much more funds.

Recommendation

Do not allow arbitrary paths.

3S-MORE-H01

Market interest is not always accrued

Id	3S-MORE-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #18ede5e.

Description

The **`market`** accrues interest, which changes **`totalAssets()`** in the**`LoopStrategy`**. Thus, everytime there is some action that modifies the amount of assets, interest should always be accrued first. However, **`markets.accrueInterest()`** in only called on **`_withdraw()`**, post shares/assets calculation, which will lead to an incorrect asset withdrawal.

On deposit/mint, it also does not accrue market interest before calculating the shares to receive.

In the view functions, such as **`totalAssets()`**, **`expectedAmountsToWithdraw()`** or **`maxWithdraw()`**, it also does not consider the accrued interest from the market.

Recommendation

Accrue market interest before each action and ideally find a way to fetch the accrued interest in a view function (the market does not seem to have this ability currently).

3S-MORE-M01

`Vault` cap is not considered in the **`maxDeposit()`** calculations, which may make deposits fail

Id	3S-MORE-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #4793c1c.

Description

`LoopStrategy::maxDeposit()` does not consider the vault cap, which limits the amount that can be deposited if the utilization rate is close to the **`targetUtilization`**. Hence, deposits will revert.

Recommendation

Consider incorporating the **`maxDeposit()`** of the vault.

3S-MORE-M02

`LoopStrategy` is vulnerable to inflation attacks

Id	3S-MORE-M02
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Acknowledged.

Description

`LoopStrategy` inherits **`ERC4626Upgradeable`**, which calculates shares deposited as (**`_decimalsOffset()`** is 0):

```
shares = assets * (totalSupply() + 1) / (totalAssets() + 1);
```

As such, the following attack is possible:

- User deposits 1e18 assets.

- Attacker frontruns and deposits 1 asset, minting **`1 * 1 / 1 = 1`** share.

- Attacker donates **`2e18`** worth of vault shares to the market.

- User1 deposits 1e18 assets and receives no shares, **`shares = 1e18 * 2 / (2e18 + 1) = 0`**.

- User2 deposits 1.5e18 assets and receives no shares, **`shares = 1.5e18 * 2 / (3e18 + 1) = 0`**.

- Attacker redeems **`1 * (4.5e18 + 1) / (1+1) = 2.25e18`**.

Recommendation

Consider using a bigger decimals offset or do an initial deposit.

3S-MORE-L01

In case of high **`ltv`** and unfavourable swap, **`onMoreFlashLoan()`** may underflow

Id	3S-MORE-L01
Classification	Low
Severity	Low
Likelihood	
Category	Bug
Status	Acknowledged.

Description

`LoopStrategy::onMoreFlashLoan()` underflows when **`cost > collateralToWithdraw`**, which may happen in case the market has a high **`ltv`** and significant slippage occurs.

Recommendation

Consider using the **`wFlow`** from the **`vault`** in this case.

3S-MORE-L02

`LoopStrategy::totalAssets()` tracks the **`wFlow`** balance, but this is not redeemable

Id	3S-MORE-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #bc94096.

Description

The **`LoopStrategy`** is not supposed to hold **`wFlow`** directly, but this is still tracked in **`totalAssets()`**. However, this is not redeemed in **`_withdraw()`**, so it will be stuck.

Recommendation

Consider withdrawing **`wFlow`** on **`_withdraw()`** or remove the balance from the **`totalAssets()`** calculation.

3S-MORE-N01

In **`LoopStrategy::withdraw()`**, **`_updateLastTotalAssets()`** is called after **`_withdraw()`**

Id	3S-MORE-N01
Classification	None
Category	Bug
Status	Addressed in #c4b4f1a.

Description

In **`LoopStrategy::withdraw()`**, **`_updateLastTotalAssets()`** is called after **`_withdraw()`**.

Recommendation

Follow the CEI pattern and update the total assets before the **`_withdraw()`** call.

3S-MORE-N02

> should be used instead of `!=` in `LoopStrategy::_deposit()`

Id	3S-MORE-N02
Classification	None
Category	Suggestion
Status	Addressed in #a1ae67f.

Description

`LoopStrategy::_deposit()` stops iterating if the utilization ratio reaches exactly the target, that is, `market.totalSupplyAssets.wMulDown(targetUtilization) == totalBorrowAssets`. However, if the utilization is bigger, it continues.

Recommendation

This should not be possible given the rest of the function, but using `>` is more correct.

3S-MORE-N03

Typo in **`LoopStrategy::_maxDeposit()`**

Id	3S-MORE-N03
Classification	None
Category	Good Code Practices
Status	Addressed in #40132e8.

Description

`LoopStrategy::_maxDeposit()` has as a comment **`totolSuppliable`**, which has a typo.

Recommendation

Replace **`totolSuppliable`** with **`totalSuppliable`**.

3S-MORE-N04

Approval is not reset to 0 after approving the **`markets`** for **`wFlow assets`** repayment in **`LoopStrategy::onMoreFlashLoan()`**

Id	3S-MORE-N04
Classification	None
Category	Suggestion
Status	Addressed in #f17f609.

Description

`LoopStrategy::onMoreFlashLoan()` does not reset the approval of the **`markets`** to 0 after approving the repayment of **`wFlow`**.

Recommendation

This is not a problem for **`wFlow`**, but some tokens revert when approving from a null amount to a non null amount, so consider resetting the approval.

3S-MORE-N05

`safeApprove()` should be used instead of `approve()`

Id	3S-MORE-N05
Classification	None
Category	Suggestion
Status	Addressed in #72adb3b.

Description

Some tokens do not return a bool on approval, so `safeApprove()` should be used.

Recommendation

Consider replacing all instances of `approve()` by `safeApprove()`. Can be kept as is as the current tokens work without issues.

3S-MORE-N06

`paramsOfTheMarket` is unused in the `LoopStrategy`

Id	3S-MORE-N06
Classification	None
Category	Good Code Practices
Status	Addressed in #50a9248.

Description

`paramsOfTheMarket` is unused in the `LoopStrategy`.

Recommendation

Remove or deprecate the variable (comment or rename it).