Three Sigma

Code Audit

M^0

M^0 Institutional $M generation

Disclaimer

Code Audit

M^0 Institutional $M generation

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

M^0 Institutional $M generation

Table of Contents

Summary

Code Audit

M^O Institutional $M generation

Summary

Three Sigma audited M^0 in a 2 day engagement. The audit was conducted from 21-11-2024 to 22-11-2024.

Protocol Description

UsualM is an extension built on top of Smart $M. The exchange rate between Smart $M and UsualM is 1:1. All Smart $M locked in the UsualM extension, serving as backing collateral for UsualM, will earn yield once UsualM is added to the earners' list by M^0 governance. This yield will be automatically transferred to the yield claim recipient designated by the extension partner. Additionally, UsualM includes pausing and permissioned role-controlled functionalities unavailable in Smart $M.

Scope

Code Audit

M^0 Institutional $M generation

Scope

UsualM.sol

NAVProxyMPriceFeed.sol

Assumptions

OpenZeppelin is considered secure.

Code Maturity Evaluation

Code Audit

M^0 Institutional $M generation

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

M^0 Institutional $M generation

Project Dashboard

Application Summary

Name	M^0
Commit	7ed8340dcfdca31ae24c4d2ceca3bebb47798b04
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	21-11-2024 to 22-11-2024
№ of Auditors	1
Review Time	2 days

Vulnerability Summary

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

None	1	1	0

Category Breakdown

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

Code Maturity Evaluation

Code Audit

M^0 Institutional $M generation

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. All access control is correctly implemented.
Arithmetic	Satisfactory. No rounding errors were found.
Centralization	Satisfactory. No significant points of centralization were found.
Code Stability	Satisfactory. The code was stable throughout the audit.
Upgradeability	Satisfactory. The contracts are upgradeable.
Function Composition	Satisfactory. The code was correctly split into helper functions.
Front-Running	Moderate. One front-running issue was found.
Monitoring	Satisfactory. Most events are emitted.
Specification	Satisfactory. The code follows the specifications.
Testing and Verification	Satisfactory. The codebase implements unit and fuzz tests.

Findings

Code Audit

M^0 Institutional $M generation

Findings

3S-M^0-L01

`UsualM::wrapWithPermit()` may be DoSed by frontrunning permit

Id	3S-M^0-L01
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #4c5b666.

Description

`UsualM::wrapWithPermit()` calls **`mToken::permit()`**, which may revert if the signature was already spent. An attacker can frontrun this call and spend the signature themselves directly in the **`mToken`** contract, making the **`UsualM::wrapWithPermit()`** call revert.

Recommendation

Consider wrapping the permit call in a try/catch block.

3S-M^0-N01

`UsualM::_update()` could specify the function to call instead of using `super()`

Id	3S-M^0-N01
Classification	None
Category	Good Code Practices
Status	Addressed in #4c5b666.

Description

`UsualM::_update()` calls `super._update(from, to, amount);`, but overrides `ERC20PausableUpgradeable` and `ERC20Upgradeable`. It works correctly due to the inheritance chain, but if the order of inheritance was different, it could call `ERC20Upgradeable._update()`, which does not have the pausable checks.

Recommendation

Consider explicitly calling `ERC20PausableUpgradeable._update(from, to, amount);`.

```
function _update(
    address from,
    address to,
    uint256 amount
) internal virtual override(ERC20PausableUpgradeable, ERC20Upgradeable) {
    UsualMStorageV0 storage $ = _usualMStorageV0();
    if ($.isBlacklisted[from] || $.isBlacklisted[to]) revert Blacklisted();
    ERC20PausableUpgradeable._update(from, to, amount);
}
```