Blackthorn

Security Review For **Usual Labs**

Collaborative Audit Prepared For:	**Usual Labs**
Lead Security Expert(s):	**xiaoming90**
	bughuntoor
	0x37
	nirohgo
Date Audited:	**December 9th - December 11th**
Final Commit:	**aa41e2ccf41c393b7746a114c0dee3b99222e5ac**

Introduction

Usual is a secure and decentralized fiat-backed stablecoin issuer that redistributes value and ownership through the USUALtoken.
Take control, make an impact, and grow with us.

Scope

Repository: https://github.com/mellow-finance/mellow-alm-toolkit

Commit: 1fb98689616496bdf843e573232cfed442df95b0

Contracts:

- src/Core.sol
- src/libraries/PositionLibrary.sol
- src/libraries/PositionValue.sol
- src/modules/strategies/PulseStrategyModule.sol
- src/modules/velo/VeloAmmModule.sol
- src/modules/velo/VeloDepositWithdrawModule.sol
- src/oracles/VeloOracle.sol
- src/utils/DefaultAccessControl.sol
- src/utils/LpWrapper.sol
- src/VeloDeployFactory.sol
- src/VeloFarm.sol

Final Commit Hash

https://github.com/mellow-finance/mellow-alm-toolkit/commit/aa41e2ccf41c393b7746a114c0dee3b99222e5ac

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	1	3

Issues Not Fixed or Acknowledged

High	Medium	Low/Info
0	0	0

Security Experts Dedicated to This Review

@xiaoming90

[illegible]

@bughuntoor

[illegible]

@0x37

[illegible]

@nirohgo

[illegible]

Issue M-1: A change of wrappedM's implementation would allow for bypassing of the `mintCap`

Source: https://github.com/sherlock-audit/2024-12-usualm/issues/13

Summary

Vulnerability Detail

When wrapping `wrappedM` into `UsualM`, the contract has introduced a `mintCap`. Ideally. it is supposed to set a cap of the damage that can be done to the Usual protocol, had the `wrappedM` been hacked/ M0's multi-sig gets compromised.

```
function wrap(address recipient, uint256 amount) external returns (uint256) {
    if (amount == 0) revert InvalidAmount();

    return _wrap(msg.sender, recipient, amount);
}
```

```
function _wrap(address account, address recipient, uint256 amount) internal returns
↪   (uint256 wrapped) {
    UsualMStorageV0 storage $ = _usualMStorageV0();

    // Check if the new total supply would exceed the mint cap.
    if (totalSupply() + amount > $.mintCap) revert MintCapExceeded();

    // NOTE: The behavior of `IWrappedMLike.transferFrom` is known, so its return
↪   can be ignored.
    IWrappedMLike($.wrappedM).transferFrom(account, address(this), amount);

    _mint(recipient, wrapped = amount);
}
```

However, as the current implementation lacks a `nonReentrant` guard and does to `mintCap` check early, this still leaves the contract vulnerable to infinite minting.

If the M0 multi-sig goes rogue, they can update the implementation of the contract, to one that has a hook/ custom logic on its `transferFrom` method. This logic would re-enter wrap, as many times as they wish. Since the `mintCap` check is done before the external call, this would allow them to bypass the cap.

As they'll be able to mint infinite tokens, they'll then be able to exchange them for USD0/ USYC, via the `daoCollateral` contract.

Impact

Drain of the whole protocol

Code Snippet

https://github.com/sherlock-audit/2024-12-usualm/blob/ed27dba12c620e907b166ee77fc5627ffa01f3d2/UsualM.sol#L253

Tool Used

Manual Review

Recommendation

Instead of checking the `mintCap` within `_wrap`, check it within `_update` whenever `from == address(0)`. This would also serve as a safeguard in case the contract gets upgraded and more logic gets added.

Issue L-1: `NAVProxyMPriceFeed` has wrongly hardcoded its decimals to 8.

Source: https://github.com/sherlock-audit/2024-12-usualm/issues/14

Summary

Vulnerability Detail

If we look at the code of `NAVProxyMPriceFeed`, we'll see that it has both hardcoded its decimals to 8 and expects Chainlink to always return its price in 8.

```
/// @notice NAV price threshold that defines 1$ M price.
int256 public constant NAV_POSITIVE_THRESHOLD = 1e8;
```

```
function decimals() public pure returns (uint8) {
    return 8;
}
```

However, if we look at the code of the Chainlink aggregator (which serves as a wrapper to the actual aggregator reporting the price), we'll see that the `decimals` are not hardcoded, but rather fetched from current actual aggregator.

```
function decimals()
  external
  view
  override
  returns (uint8)
{
  return currentPhase.aggregator.decimals();
}
```

Note that the aggregator is subject to change. From example, the ETH/ USDC aggregator has been changed 7 times in its lifetime. I couldn't find any example of changing decimals between aggregator versions, but since it is theoretically possible, it is advised that `NAVProxyMPriceFeed` also dynamically fetches the decimals.

Impact

Wrong prices returned.

Code Snippet

https://github.com/sherlock-audit/2024-12-usualm/blob/b86b68f7f401f2555c034fb928f501c6a5693072/NAVProxyMPriceFeed.sol#L35

Tool Used

Manual Review

Recommendation

Dynamically fetch the decimals

Issue L-2: WrappedM's possible attack vectors

Source: https://github.com/sherlock-audit/2024-12-usualm/issues/12

Summary

Vulnerability Detail

Following is the analysis of how the WrappedM could be used to attack UsualM.

Attack Vector 1:

Admin (MZero's TTG governance) of the WrappedM contract can upgrade the Wrapped M contract to mint infinite wM tokens and wrap them in UsualM to use as collateral in the Usual protocol.

Attack Vector 2:

Whenever there is any transfer of wM tokens in and out of the `UsualM` contract (mainly during wrap and unwrap), the `WrappedMToken._claim()` below will be triggered, and the yield will be forwarded to Usual Treasury.

Thus, the admin (MZero's TTG governance) of the WrappedM contract can DOS the `UsualM` contract by setting the `claimRecipient_` to an address with a long chain of recipients (this is a known issue per the comment below and WrappedM's audit-report).

If this happens, no one can wrap or unwrap the UsualM.

```
function _claim(address account_, uint128 currentIndex_) internal returns (uint240
↪ yield_) {
..SNIP..
    address claimOverrideRecipient_ = claimOverrideRecipientFor(account_);
    address claimRecipient_ = claimOverrideRecipient_ == address(0) ? account_ :
↪ claimOverrideRecipient_;
..SNIP..
    if (claimRecipient_ != account_) {
        // NOTE: Watch out for a long chain of earning claim override recipients.
        _transfer(account_, claimRecipient_, yield_, currentIndex_);
    }
}
```

Attack Vector 3:

WrappedM contract governance can upgrade the WrappedM contract with custom _transfer code that transfers all or part of WrappedM sent to UsualM to another address. Impact: UsualM not fully backed by WrappedM (some unwraps would fail), and indirectly, USD0 will be undercollateralized.

Impact

Malicious admin of WrappedM contract can affect the Usual protocol negatively in the following manner:

1. Mint unlimited UsualM/USD0
2. Cause DOS to wrapping/unwrapping of UsualM
3. Steal a portion of WrappedM transferred to the Usual protocol
4. Cause USD0 to be undercollateralized

Code Snippet

Tool Used

Manual Review

Recommendation

- Implement a minting cap to limit losses (already implemented)
- Set up a monitoring system to automatically pause the relevant contract if certain invariants of the UsualM are broken. For instance,
    - UsualM's `totalSupply()` should not exceed the minting cap.

- Implement rate limiting. Restrict the number of UsualM tokens that can be minted within a certain period of time (e.g., 100K tokens per hour)
- Setup notification to alert Usual's protocol team whenever the `WrappedM` contract is upgraded. Upon receiving the notification, the protocol team should check with the M0 team on the rationale behind the upgrade, check out for any announcement regarding the upgrade OR review the changelogs (if any). This is to ensure that the upgrade is a normal business operation.
- The actual admin of WrappedM is controlled by the M^ZERO Two Token Governance (TTG). All proposals will be submitted and voted on in the current epoch, and the passed proposal will be executed in the subsequent epoch. Consider monitoring the proposals submitted to ensure that no malicious upgrade to the `WrappedM` contract will be performed that might harm the Usual protocol.

Issue L-3: UsualM's oracle configuration

Source: https://github.com/sherlock-audit/2024-12-usualm/issues/11

Summary

Outdated prices might be used within the protocol due to the timeout configured.

Vulnerability Detail

Within the Chainlink's price feed, two (2) conditions will trigger an price update on-chain:

1. Deviation Threshold - Update occurs when the price moves more than the pre-defined threshold.
2. Heartbeat Threshold - Update occurs when there is no update for the pre-defined duration since the last update.

Upon inspecting the Chainlink’s M NAV price feed, which will be used to determine the price of UsualM, it was found that the heartbeat of the price feed is currently set to 26 hours (93600 seconds). This means that if 26 hours have passed since the last price update to on-chain, the oracles should trigger an update to the price regardless of the price movement.

From the audit documentation, it was found that the `timeout` for the WrappedM tokens will be planned to be set to 7 days (604800 seconds).

Following an attempt to perform analysis on the timeout configured on all the existing and upcoming RWA's ClassicalOracle:

RWA	Chainlink's Heartbeat	ClassicalOracle's Timeout Configured & Buffer	Buffer	Remarks
USDC	24 hours (86400 seconds)	24.16 hours (87000 seconds)	10 minutes (600 seconds)	OK. 10-minute (or 50-blocks) buffer for the oracles to start the price update, factoring in potential network congestion.
USYC	N/A (Since price is updated/re-ported manually by Hashnote entity. Not using Chainlink service. Oracle is owned by Hashnote).	4 days (345600 seconds)	N/A	OK. The 4 days are to factor in scenarios like bank holidays or catastrophic events as per feedback by the Usual's protocol team.
UsualM	26 hours (93600 seconds)	7 days (604800 seconds)	~6 days (511200 seconds)	Buffer might be too long. 5.9 days (or 42600 blocks) buffer for the oracles to start the price update.

If the price is not updated with the heartbeat period, the price is considered stale. However, given the oracles will only start ”refreshing” or update the price once the heartbeat period is over, an additional buffer needs to be considered to factor in potential delays, such as:

1. It takes some time for the oracles to notice the heartbeat period is over and to kick-start the update process.
2. Some network congestion resulted in the update transaction being pending.

The buffer ensures that the price is not pre-maturely marked as stale due to the above condition, resulting in unnecessary revert, which might break some of the protocol features.

The 10 minutes buffer in the USDC's ClassicalOracle seems reasonable. However, the ~6 days buffer in the upcoming UsualM's ClassicalOracle appears excessive. It should not take ~6 days for the oracles to prepare and submit a price update to the Chainlink service after the heartbeat period is over.

If the price update does not occur within a reasonable timeframe after the heartbeat period is over, it might indicate that there are more serious events going on (e.g., oracle infrastructure or network is down OR Chainlink's price update service not working). In this case, it would be prudent not to use the price from Chainlink's price feed in such a scenario because the price is likely to be stale or outdated.

Impact

Outdated prices might be used within the protocol.

Code Snippet

N/A. Configuration issue.

Tool Used

Manual Review

Recommendation

Consider setting the `timeout` in the upcoming Usual's ClassicalOracle to be the sum of the heartbeat (26 hours) plus a buffer (e.g., 10/20/30 minutes or reasonable duration needed for the oracle to execute an update).

Disclaimers

Blackthorn does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.