# Usual M extensions

## Security Review

Cantina Managed review by:
**Desmond Ho**, Lead Security Researcher
**Akshay Srivastav**, Security Researcher

January 28, 2025

# Contents

# 1 Introduction

## 1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

## 1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

## 1.3 Risk assessment

| Severity | Description |
|---|---|
| **Critical** | *Must* fix as soon as possible (if already deployed). |
| **High** | Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users. |
| **Medium** | Global losses <10% or losses to only a subset of users, but still unacceptable. |
| **Low** | Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies. |
| **Gas Optimization** | Suggestions around gas saving practices. |
| **Informational** | Suggestions around best practices or readability. |

### 1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

# 2 Security Review Summary

Usual is a Stablecoin DeFi protocol that redistributes control and redefines value sharing. It empowers users by aligning their interests with the platform's success.

$USD0 is a USUAL native stablecoin with real-time transparency of reserves, fully collateralized by US Treasury Bills. This eliminates fractional reserve risks and protects against the bankruptcy risks of fiat-backed stablecoins.

$USD0 can be locked into $USD0++, a liquid 4-year bond backed 1:1, offering users the alpha-yield distributed as points and ensuring at least the native yield of their collateral. This provides enhanced stability and attractive returns for holders.

From Dec 13th to Dec 15th the Cantina team conducted a review of m-extensions on commit hash bb6be6e0. The team identified a total of **7** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 1
- Gas Optimizations: 1
- Informational: 5

# 3 Findings

## 3.1 Low Risk

### 3.1.1 Blacklisted account can perform `UsualM` token transfers from other non-blacklisted accounts

**Severity:** Low Risk

**Context:** UsualM.sol#L281

**Description:** The `UsualM._update` function implements blacklist checks for sender and receiver of `UsualM` tokens. This is done to prevent transfers of tokens from/to a blacklisted account.

However the implementation does not prevent a blacklisted account from transferring tokens from a non-blacklisted account via the `transferFrom` function.

**Scenario:**

- Suppose a vulnerability is found in an orderbook DEX that directly `transferFromS` `UsualM` tokens from makers to takers and vice versa.
- The `BLACKLIST_ROLE` blacklists the DEX contract to protect `UsualM` token holders from attack.
- But due to the current implementation the blacklisted DEX contract can still transfer `UsualM` tokens from user accounts.

**Recommendation:** Add this change:

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

      // Check if minting would exceed the mint cap
      if (from == address(0) && totalSupply() + amount > $.mintCap) revert MintCapExceeded();

      ERC20PausableUpgradeable._update(from, to, amount);
  }
```

**Usual:** Not acknowledged.

## 3.2 Gas Optimization

### 3.2.1 `newMintCap` can be defined as `uint96` to avoid safecasting

**Severity:** Gas Optimization

**Context:** UsualM.sol#L128

**Description:** `newMintCap` is defined as type `uint256`, then safecasted to `uint96`. The safecasting becomes redundant if `newMintCap` was directly defined as `uint96`.

**Recommendation:**

```
- function setMintCap(uint256 newMintCap) external {
+ function setMintCap(uint96 newMintCap) external {
```

**Usual:** Acknowledged.

**Cantina Managed:** Acknowledged.

## 3.3 Informational

### 3.3.1 Risks and safeguards for wrappedM integration

**Severity:** Informational

**Context:** *(No context files were provided by the reviewer)*

**Description:** One of the requests of the security review was to "*document and analyze how wrappedM could be used to attack usualM & what safeguards usualM can consider/has implemented against it*". Our findings of the non-exhaustive risks and safeguards in place are as follows:

**Risks:**

- WrappedM proxy upgrading to a malicious implementation.
- Yield redirection, where the yield recipient of wrappedMs held by the `UsualM` contract can be set by M0 to other addresses that aren't the treasury. A possible case could be that the yield recipient of `UsualM` is set as the `UsualM` address itself. In this case, the contract will accrue WrappedM token balance which cannot be recovered.
- Unauthorised minting of M0 that's subsequently wrapped for `UsualM` and then used for minting USD0 while NAV price remains high.

**Safeguards:**

- Global mutable minting cap.
- Pausing transfer functionality on a global level.
- Blacklists to halt transfers for specific accounts.
- Permissioned unwrapping of `UsualM` back to `wrappedM`.

One additional safeguard that could be implemented is the introduction of delays for larger volume mints. For instance, up to 5M `UsualM` can be minted instantly per day, any excess will require manual approval by an authorized party and / or a delay period.

**Usual:** Acknowledged.

**Cantina Managed:** Acknowledged.

### 3.3.2 Theoretical `wrappedM` supply could exceed minting cap

**Severity:** Informational

**Context:** UsualM.sol#L39

**Description:** The minting cap is stored as `uint96` to be tightly packed with the `wrappedM` address, while the balances and accrued yield (and thus total supply) for `wrappedM` are stored as `uint240`. It is highly unlikely for `wrappedM`'s total supply to practically exceed $2^{96}$, but it is something to be aware of.

**Recommendation:** No immediate action is necessary, but it may be worthwhile adding a comment to acknowledge the limitation of using a smaller variable type.

**Usual:** Acknowledged.

**Cantina Managed:** Acknowledged.

### 3.3.3 Support ERC1271 smart contract permits

**Severity:** Informational

**Context:** UsualM.sol#L108

**Description:** `wrappedM` has another `permit()` function that takes in `bytes memory` signature instead of the `(v, r, s)` tuple to allow smart contract wallets to verify signatures and authorize approvals, as per ERC1271. Currently, this functionality is not supported in the implementation, which would limit compatibility with smart contract wallets.

**Recommendation:** Consider overloading the `permit` function to support smart contract permits.

```
function wrapWithPermit(
   address recipient,
   uint256 amount,
   uint256 deadline,
   bytes calldata signature
)
```

**Usual:** Acknowledged.

**Cantina Managed:** Acknowledged.

### 3.3.4 Comment improvements

**Severity:** Informational

**Context:** AggregatorV3Interface.sol#L7-L8, NAVProxyMPriceFeed.sol#L38-L41, IUsualM.sol#L117, UsualM.sol#L270

**Description:** List of issues:

1. `IUsualM.sol#L117`: Misspelling, `deciamls` ⇒ `decimals`.
2. `UsualM.sol#L270`: Add documentation about `mintCap` and `paused` status checks.
3. `NAVProxyMPriceFeed.sol#L38`: Add a custom comment as the function no longer returns the "*number of decimals used by the aggregator*" as mentioned in the doc of `AggregatorV3Interface.decimals` function.

**Usual:** Acknowledged.

**Cantina Managed:** Acknowledged.

### 3.3.5 Wrap operation breaks the CEI pattern

**Severity:** Informational

**Context:** UsualM.sol#L246-L253

**Description:** The `UsualM._wrap` function performs an external call to the `WrappedM` token contract before performing the internal `_mint` operation. The widely used `checks-effects-interactions` pattern suggests to always perform internal state updates before making an external call. Breaking this pattern can lead to reentrancy issues.

**Recommendation:** Consider performing the `_mint` operation before the external call. Or add a `nonReentrant` modifier to `wrap` and `wrapWithPermit` functions.

**Usual:** Acknowledged, but no impact. Will be remedied in a future contract upgrade.

**Cantina Managed:** Acknowledged.