hexens x USUAL

Security Review Report for Usual

June 2025

Table of Contents

1. About Hexens

Hexens is a pioneering cybersecurity firm dedicated to establishing robust security standards for Web3 infrastructure, driving secure mass adoption through innovative protection technology and frameworks. As an industry elite experts in blockchain security, we deliver comprehensive audit solutions across specialized domains, including infrastructure security, Zero Knowledge Proof, novel cryptography, DeFi protocols, and NFTs.

Our methodology combines industry-standard security practices combined with unique methodology of two teams per audit, continuously advancing the field of Web3 security. This innovative approach has earned us recognition from industry leaders.

Since our founding in 2021, we have built an exceptional portfolio of enterprise clients, including major blockchain ecosystems and Web3 platforms.

2. Executive Summary

This report covered the audit of the UsualXLockup contract of the Usual DAO contracts repository. The contracts allows users to lock USUAL for a certain period of time.

Our security assessment was a full review of the new smart contracts in scope, spanning a total of 1 week.

During our audit, we did not identify any major severity vulnerabilities.

We did identify several minor severity vulnerabilities and code optimisations.

All of our reported issues were fixed or acknowledged by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

3. Security Review Details

- **Review Led by**

Kasper Zwijsen, Head of Audits
Jahyun Koo, Lead Security Researcher

- **Scope**

The analyzed resources are located on:

🔗 https://github.com/usual-dao/core-protocol
  - Limited to UsualXLockup.sol and UsualX.sol

📌 Commit: `f497c2e7d53786860d02d702be7342a4b0c2695c`

The issues described in this report were fixed in the following commits:

🔗 https://github.com/usual-dao/core-protocol/tree/

📌 (PR 30) Commit: `9f1d02ea2953c554f0c1abacefdc43fe0cc0c45b`

📌 (PR 34) Commit: `17dfb4aabd481398f4f2c8ab4d83fd67603dab46`

- **Changelog**

30 May 2025	Audit start
05 June 2025	Initial report
07 June 2025	Revision received
10 June 2025	Final report

4. Severity Structure

The vulnerability severity is calculated based on two components:

1. Impact of the vulnerability
2. Probability of the vulnerability

Impact	Probability			
	Rare	Unlikely	Likely	Very likely
Low	Low	Low	Medium	Medium
Medium	Low	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

▪ Severity Characteristics

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical	Vulnerabilities that are highly likely to be exploited and can lead to catastrophic outcomes, such as total loss of protocol funds, unauthorized governance control, or permanent disruption of contract functionality.
High	Vulnerabilities that are likely to be exploited and can cause significant financial losses or severe operational disruptions, such as partial fund theft or temporary asset freezing.

Medium	Vulnerabilities that may be exploited under specific conditions and result in moderate harm, such as operational disruptions or limited financial impact without direct profit to the attacker.
Low	Vulnerabilities with low exploitation likelihood or minimal impact, affecting usability or efficiency but posing no significant security risk.
Informational	Issues that do not pose an immediate security risk but are relevant to best practices, code quality, or potential optimizations.

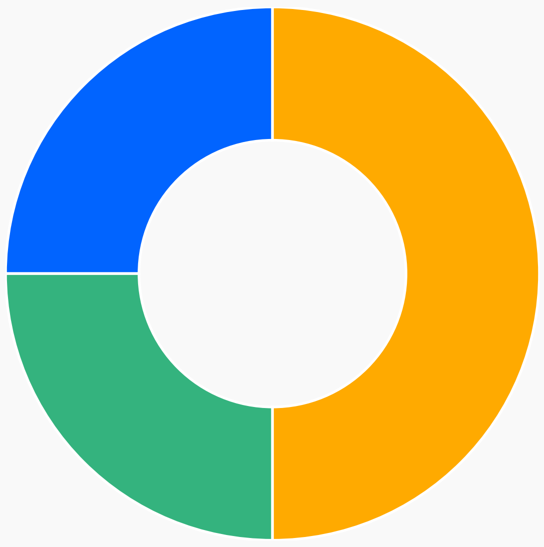
▪ Issue Symbolic Codes

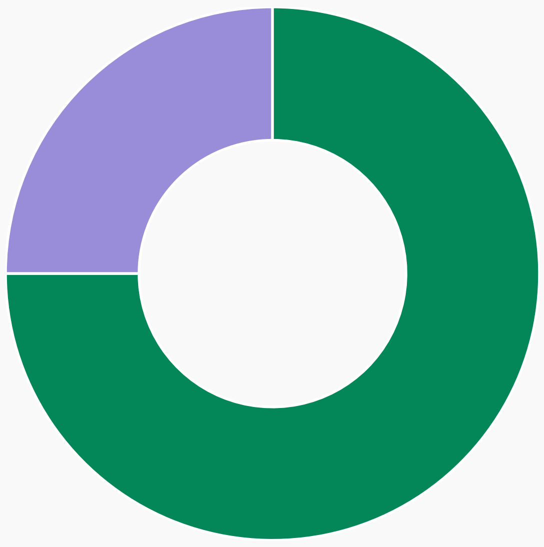
Each identified and validated issue is assigned a unique symbolic code during the security research stage.

Due to the structure of the vulnerability reporting flow, some rejected issues may be missing.

5. Findings Summary

Severity	Number of findings
🟥 Critical	0
🟥 High	0
🟧 Medium	2
🟩 Low	1
🟦 Informational	1
Total:	**4**





6. Weaknesses

This section contains the list of discovered weaknesses.

USUAL1-1 | releaseAndLock Function Bypasses Protocol-Enabled Lock Durations

Fixed ✓

Severity:	Medium	Probability:	Unlikely	Impact:	Medium

Path:

UsualXLockup.sol#L290-L307

Description:

When users create a new lock using the **lock()** function, the contract enforces that the **lockDuration** matches one of the durations enabled by the protocol (e.g., 1 month, 3 months, 6 months, or 1 year).

```
    function lock(address receiver, uint256 amount, uint256 lockDuration)
      external
      nonReentrant
      whenNotPaused
  {
  --- SNIP ----
      UsualXLockupStorageV0 storage $ = _usualXLockupStorageV0();
      if ($.lockDurations[lockDuration] == false) {
          revert LockDurationIsNotEnabled();
      }

      _lock($, msg.sender, receiver, amount, lockDuration);
  }
```

However, when a user uses the **releaseAndLock()** function to release an expired lock and creating a new one, this check is missing:

```
    function releaseAndLock(
        address user,
        uint256 positionIndex,
        uint256 amount,
        uint256 lockDuration
    ) external nonReentrant whenNotPaused {
        if (user == address(0)) {
            revert NullAddress();
        }
        if (amount == 0) {
            revert AmountIsZero();
        }
        UsualXLockupStorageV0 storage $ = _usualXLockupStorageV0();
        _release($, user, positionIndex);
        _lock($, msg.sender, user, amount, lockDuration);
    }
```

As a result, a user could bypass the protocol's enabled lock durations by passing in any arbitrary **lockDuration**.

This could allow a user to create a lock with 0 or 1 second duration and still use the top-up function after the lock has expired.

Remediation:

Consider adding the following check to prevent arbitrary lock durations.

```
function releaseAndLock(
    address user,
    uint256 positionIndex,
    uint256 amount,
    uint256 lockDuration
) external nonReentrant whenNotPaused {
    if (user == address(0)) {
        revert NullAddress();
    }
    if (amount == 0) {
        revert AmountIsZero();
    }
    UsualXLockupStorageV0 storage $ = _usualXLockupStorageV0();

++    if ($.lockDurations[lockDuration] == false) {
++        revert LockDurationIsNotEnabled();
++    }

    _release($, user, positionIndex);
    _lock($, msg.sender, user, amount, lockDuration);
}
```

USUAL1-6 | UsualXLockup clearUsersPositions can be blocked by user

Fixed ✓

Severity:	Medium	Probability:	Likely	Impact:	Medium

Path:

`UsualXLockup.sol:clearUsersPositions#L245-L262`

Description:

The function **clearUsersPositions** can be used by the admin to instantly clear all positions of a user. It will reset the user's total amount to 0 and use **delete** on the array of positions of the user.

However, the usage of **delete** on an array is not of constant gas cost. On the contrary, it scales linearly with the number of elements in the array. Each **LockPosition** is 4 slots wide and so the call to **clearUsersPositions** will clear 4 cold slots for each element.

By having one real position and a few thousand dummy positions of 1 Wei, the user can block the call to **clearUsersPositions** by the admin, as the delete on line 258 would revert with out-of-gas:

```
delete $.userLockPositions[user];
```

```
    function clearUsersPositions(address[] memory users) external nonReentrant
{
        UsualXLockupStorageV0 storage $ = _usualXLockupStorageV0();
        $.registryAccess.onlyMatchingRole(POSITION_UNLOCKER_ROLE);
        for (uint256 i; i < users.length; i++) {
            address user = users[i];
            if (user == address(0)) {
                revert NullAddress();
            }
            uint256 amount = $.userLockedAmount[user];
            if (amount == 0) {
                revert AmountIsZero();
            }
            $.userLockedAmount[user] = 0;
            delete $.userLockPositions[user];
            $.usualX.safeTransfer(user, amount);
        }
        emit AllPositionsCleared(users);
    }
```

Remediation:

We would recommend to have another variant of the clear function iterate over a given number of elements before terminating, instead of deleting everything at once. $.userLockedAmount[user] should be decreased accordingly.

USUAL1-2 | Blacklisted users positions can't be cleared and potentially still accrue rewards

Acknowledged

Severity:	Low	Probability:	Unlikely	Impact:	Low

Path:

UsualXLockup.sol#L245-L262

Description:

When a user is blacklisted while having an active lock in **UsualXLockup**, there might be the need to clear it's position, the person with the **POSITION_UNLOCKER_ROLE** would call the function **clearUsersPositions**:

```
    function clearUsersPositions(address[] memory users) external nonReentrant
{
        UsualXLockupStorageV0 storage $ = _usualXLockupStorageV0();
        $.registryAccess.onlyMatchingRole(POSITION_UNLOCKER_ROLE);
        for (uint256 i; i < users.length; i++) {
            address user = users[i];
            if (user == address(0)) {
                revert NullAddress();
            }
            uint256 amount = $.userLockedAmount[user];
            if (amount == 0) {
                revert AmountIsZero();
            }

            //@audit should ther ebe a check if the otal amount is
            $.userLockedAmount[user] = 0;
            delete $.userLockPositions[user];
            $.usualX.safeTransfer(user, amount);
        }
        emit AllPositionsCleared(users);
    }
```

However, there's a problem with the transfer part. The **UsualX** token checks if either the sender or receiver is blacklisted during a transfer using the **_update** function:

```
    function _update(address from, address to, uint256 amount)
         internal
         override(ERC20Upgradeable)
     {
         UsualXStorageV0 storage $ = _usualXStorageV0();
         if ($.isBlacklisted[from] || $.isBlacklisted[to]) {
             revert Blacklisted();
         }
         super._update(from, to, amount);
     }
```

Since the user is blacklisted, the transfer in **clearUsersPositions** will fail due to the blacklist check in **_update**.

Remediation:

Integrate a way to clear a position while a user is blacklisted. It should be a suitable solution for temporary blacklist as well.

Commentary from the client:

"This will be processed off-chain."

USUAL1-7 | Incorrect nat-spec comment of UsualXLockup contract top-up period

Fixed ✓

Severity:	Informational	Probability:	Rare	Impact:	Informational

Path:

UsualXLockup.sol#L44-L49

Description:

The notice above the contract says users can top up existing positions within 24 hours, but that's not aligned with the functionality, users can only top up until the end of the UTC calendar day on which the position was created, not the full 24-hours.

```
/// @title   UsualX Lockup
/// @notice  A vault contract that enables users to lock their UsualX tokens
///          for predefined durations (1, 3, 6, or 12 months). Users can create
multiple lock positions,
///          top up existing positions within 24 hours, and release tokens
after the lock period ends.
/// @author  Usual Tech team
contract UsualXLockup is PausableUpgradeable, ReentrancyGuardUpgradeable,
IUsualXLockup {
```

Remediation:

Consider changing the comment from:

top up existing positions within 24 hours

to:

top up existing positions until the end of the UTC day they were created hexens x USUAL