hexens x USUAL

Security Review Report for Usual

October 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 covers the security review of Usual sUSD0, a yield bearing vault for USD0, a synthetic accruing stablecoin denominated in USD, developed by Usual DAO.

Our security assessment was a full review of the code, spanning a total of 1 week.

During our review, 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

- **Scope**

The analyzed resources are located on:

🔗 https://github.com/usual-dao/core-protocol/pull/61

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

🔗 https://github.com/usual-dao/core-protocol/tree/9eecab78131a3d16ae13b5b5a2644bcf8b231826

- **Changelog**

20 October 2025	Audit start
24 October 2025	Initial report
29 October 2025	Revision received
3 November 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.

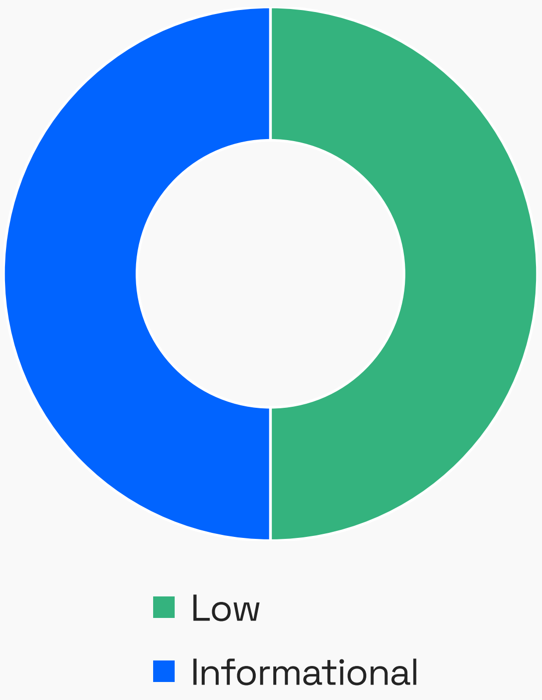
- **Issue Symbolic Codes**

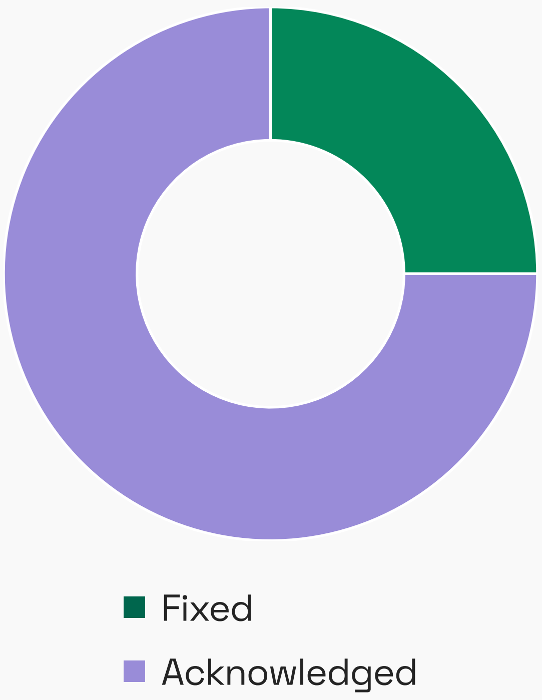
Each identified and validated issue is assigned a unique symbolic code during the security research stage.

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	0
🟩 Low	2
🟦 Informational	2
Total:	**4**





6. Weaknesses

This section contains the list of discovered weaknesses.

USL2-1 | Optimization of blacklist check in _update

Acknowledged

Severity:	Low	Probability:	Unlikely	Impact:	Low

Path:

src/vaults/SUsd0.sol:_update#L327-L337

Description:

The function **_update** is part of the **ERC20** implementation and handles the transfers. The SUsd0 contract overwrites this function to add blacklist functionality. It will check both the **from** and **to** address for blacklisted status.

The blacklist check can be skipped for **address(0)**, which will happen on a mint and a burn. One **SLOAD** would be saved every time someone deposits to or withdraws from the vault.

```
function _update(address from, address to, uint256 amount)
    internal
    override(ERC20Upgradeable)
    whenNotPaused
{
    SUsd0StorageV0 storage $ = _susd0StorageV0();
    if ($.isBlacklisted[from] || $.isBlacklisted[to]) {
        revert Blacklisted();
    }
    super._update(from, to, amount);
}
```

Remediation:

Optimize the check to skip **address(0)**, for example:

```
if ((from != address(0) && $.isBlacklisted[from])
    || (to != address(0) && $.isBlacklisted[to])) {
    revert Blacklisted();
}
```

USL2-4 | Whenever rewardClaimForAccruingDT is set, its asset token should be verified

Acknowledged

Severity:	Low	Probability:	Rare	Impact:	Low

Path:

src/modules/RevenueDistributionModule.sol#L278-L284

Description:

The **distributeAccruingDT()** function is used to distribute newly minted **usd0** tokens to the **$.rewardClaimForAccruingDT** contract by calling the **startYieldDistribution()** function after minting:

```
$.usd0.mint(address($.rewardClaimForAccruingDT), amount);
// Start yield distribution
$.rewardClaimForAccruingDT.startYieldDistribution(
    amount, block.timestamp, block.timestamp + ONE_DAY
);
```

This only works correctly when the asset of the **rewardClaimForAccruingDT** contract is **$.usd0**. Otherwise, the **startYieldDistribution()** function will revert because the incorrect token was minted.

However, there is no check to verify that the asset is correct when setting **$.rewardClaimForAccruingDT** in the **RevenueDistributionModule** in the **initialize()** function or the **setRewardClaimForAccruingDT()** function.

Remediation:

Whenever the **$.rewardClaimForAccruingDT** address is updated, its asset token should be verified to match **$.usd0**.

USL2-2 | RevenueDistributionModule missing cap check for daily accruing yield rate

Fixed ✓

Severity:	Informational	Probability:	Rare	Impact:	Low

Path:

src/modules/RevenueDistributionModule.sol:setDailyAccruingYieldRate

Description:

In the RevenueDistributionModule contract the configuration variable **dailyAccruingYieldRate** is set in the initializer from the parameter values. It is a rate expressed in micro bps (**1_000_000**) and it is correctly checked to be at most equal to the maximum value of 100%.

However, in the setter function **setDailyAccruingYieldRate**, this cap check is missing. The privileged role is able to set a rate that is higher than 100%.

```
function setDailyAccruingYieldRate(uint256 newDailyRate) external whenNotPaused {
    if (newDailyRate == 0) {
        revert AmountIsZero();
    }
    RevenueDistributionModuleStorageV0 storage $ = _revenueDistributionModuleStorageV0();
    $.registryAccess.onlyMatchingRole(OPERATOR_ADMIN_ROLE);

    if (newDailyRate == $.dailyAccruingYieldRate) {
        revert SameValue();
    }

    $.dailyAccruingYieldRate = newDailyRate;

    emit DailyAccruingYieldRateUpdated(newDailyRate);
}
```

Remediation:

We recommend to add a sanity check to **setDailyAccruingYieldRate** to not allow values greater than 100%.

USL2-3 | Redundant check for startTime in the _startYieldDistribution function

Acknowledged

Severity:	Informational	Probability:	Very likely	Impact:	Informational

Path:

src/vaults/SUsd0.sol#L343-L383

Description:

In the **_startYieldDistribution()** function, there is a check to verify that **startTime** is not below **$.periodFinish**. However, **startTime** is already required to be greater than or equal to **block.timestamp**, and there is also a check to verify that **block.timestamp** is greater than or equal to **$.periodFinish**. Therefore, **startTime** cannot be below **$.periodFinish**, making this check unnecessary.

```
function _startYieldDistribution(uint256 yieldAmount, uint256 startTime, uint256 endTime)
        internal
        override
    {
        IERC20 _asset = IERC20(asset());

        if (yieldAmount == 0) {
            revert ZeroYieldAmount();
        }
        if (startTime < block.timestamp) {
            revert StartTimeNotInFuture();
        }
        if (endTime <= startTime) {
            revert EndTimeNotAfterStartTime();
        }

        YieldDataStorage storage $ = _getYieldDataStorage();

        if (startTime < $.periodFinish) {
            revert StartTimeBeforePeriodFinish();
        }
        if (block.timestamp < $.periodFinish) {
            revert CurrentTimeBeforePeriodFinish();
        }
        _updateYield();

        uint256 periodDuration = endTime - startTime;
        uint256 newYieldRate =
            Math.mulDiv(yieldAmount, YIELD_PRECISION, periodDuration, Math.Rounding.Floor);

        if (_asset.balanceOf(address(this)) < $.totalDeposits + yieldAmount) {
            revert InsufficientAssetsForYield();
        }

        $.yieldRate = newYieldRate;
        $.periodStart = startTime;
        $.periodFinish = endTime;
        $.lastUpdateTime = startTime;
        $.isActive = true;
    }
```

Remediation:

Remove the following check:

```
if (startTime < $.periodFinish) {
    revert StartTimeBeforePeriodFinish();
}
```

hexens x USUAL