SHERLOCK

# Security Review For Usual Labs

| | |
|---|---|
| Public Best Efforts Contest Prepared For: | **Usual Labs** |
| Lead Security Expert: | **xiaoming90** |
| Date Audited: | **October 29 - November 10, 2024** |

## Introduction

Usual is a decentralized stablecoin issuer launched 3 months ago, now ranked among the top 15 with over 350MinTVLand20kholders.TheV1release, whichisthefocusofthisaudit, includestheintroductionofUSUAL, the governance token for the protocol, along with its allocation, staking and distribution logic, and the related airdrop contracts.

## Scope

Repository: usual-dao/pegasus

Branch: develop

Audited Commit: ea088fa77aac69e91ea6a191d5c11c912cd34446

Final Commit: d95c5a6a6136fe5015c217323eb093ee9de569c5

---

For the detailed scope, see the contest details.

## 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.

## Issues Found

| High | Medium |
|---|---|
| 2 | 0 |

## Issues Not Fixed or Acknowledged

| High | Medium |
|---|---|
| 0 | 0 |

## Security experts who found valid issues

KupiaSec
0xNirix
0x37
dhank
xiaoming90
0xmystery
Bigsam
zarkk01
ke1caM
LZ_security
nirohgo
PeterSR
silver_eth
Boy2000
lanrebayode77
Ragnarok
dany.armstrong90
super_jack
0xc0ffEE
LonWof-Demon
blutorque
shaflow01
uuzall

# Issue H-1: A missing reward update in UsualSP::removeOriginalAllocation will cause reduced reward accumulation for users

Source: https://github.com/sherlock-audit/2024-10-usual-labs-v1-judging/issues/16

## Found by

0x37, 0xNirix, 0xmystery, Bigsam, KupiaSec, LZ_security, PeterSR, dhank, ke1caM, nirohgo, xiaoming90, zarkk01

## Summary

The missing reward update in `removeOriginalAllocation` will cause a reduction in reward accumulation for users as the contract directly resets the `originalAllocation` variable without first updating rewards.

## Root Cause

In UsualSP.sol:376, the code in the `removeOriginalAllocation` function resets `$.originalAllocation` to zero without first updating rewards.

https://github.com/sherlock-audit/2024-10-usual-labs-v1/blob/main/pegasus/packages/solidity/src/token/UsualSP.sol#L376

```
for (uint256 i; i < recipients.length;) {
    $.originalAllocation[recipients[i]] = 0;
    $.originalClaimed[recipients[i]] = 0;

    emit RemovedOriginalAllocation(recipients[i]);
    unchecked {
        ++i;
    }
```

This causes an unintended reward loss as the `balanceOf` calculation, used for accruing rewards, will reference a reduced allocation amount on subsequent reward claims.

## Internal pre-conditions

1. `removeOriginalAllocation` is called on an address that has an active reward allocation.
2. `balanceOf` calculation for reward accrual depends on `$.originalAllocation`.

https://github.com/sherlock-audit/2024-10-usual-labs-v1/blob/main/pegasus/packages/solidity/src/token/UsualSP.sol#L443-L447

```
function balanceOf(address account) public view override returns (uint256) {
    UsualSPStorageV0 storage $ = _usualSPStorageV0();
    return
        $.liquidAllocation[account] + $.originalAllocation[account] -
 ↪  $.originalClaimed[account];
}
```

## External pre-conditions

1. The protocol is running an active reward accrual period.

## Attack Path

1. The operator calls `removeOriginalAllocation` on a recipient without first updating the reward.
2. The `originalAllocation` value for the recipient is reset to zero in storage.
3. When the recipient calls `claimReward`, the calculation in `_earned` references a reduced `balanceOf` due to `originalAllocation==0`, leading to lower rewards.

https://github.com/sherlock-audit/2024-10-usual-labs-v1/blob/main/pegasus/packages/solidity/src/modules/RewardAccrualBase.sol#L136

```
function _earned(address account) internal view virtual returns (uint256 earned) {
    RewardAccrualBaseStorageV0 storage $ = _getRewardAccrualBaseDataStorage();
    uint256 accountBalance = balanceOf(account);
    uint256 rewardDelta = $.rewardPerTokenStored -
 ↪  $.lastRewardPerTokenUsed[account];
    earned = accountBalance.mulDiv(rewardDelta, 1e24, Math.Rounding.Floor) +
 ↪  $.rewards[account]; // 1e24 for precision loss
}
```

## Impact

The users experience a reduced reward accumulation as their `originalAllocation` is zeroed out before updating rewards. This impacts the accuracy of rewards distribution and results in reward loss for affected users.

## PoC

*No response*

## Mitigation

Add `_updateReward(recipients[i])` in the `removeOriginalAllocation` function before resetting `$.originalAllocation[recipients[i]]=0`. This ensures rewards are fully accrued based on the prior allocation.

https://github.com/sherlock-audit/2024-10-usual-labs-v1/blob/main/pegasus/packages/solidity/src/token/UsualSP.sol#L366-L384

```
        for (uint256 i; i < recipients.length;) {
+            _updateReward(recipients[i]);
            $.originalAllocation[recipients[i]] = 0;
            $.originalClaimed[recipients[i]] = 0;

            emit RemovedOriginalAllocation(recipients[i]);
            unchecked {
                ++i;
            }
        }
```

## Discussion

**sherlock-admin2**

The protocol team fixed this issue in the following PRs/commits:
https://github.com/usual-dao/pegasus/pull/1729

# Issue H-2: Withdrawal fee for UsualX vault will be mis-calculated.

Source: https://github.com/sherlock-audit/2024-10-usual-labs-v1-judging/issues/34

## Found by

0x37, 0xNirix, 0xc0ffEE, 0xmystery, Bigsam, Boy2000, KupiaSec, LonWof-Demon, Ragnarok, blutorque, dany.armstrong90, dhank, lanrebayode77, shaflow01, silver_eth, super_jack, uuzall, xiaoming90, zarkk01

## Summary

The `UsualX.withdraw()` function has logic error in calculating the withdrawal fee.

## Root Cause

The UsualX.withdraw() function is following.

```
    function withdraw(uint256 assets, address receiver, address owner)
        public
        override
        whenNotPaused
        nonReentrant
        returns (uint256 shares)
    {
        UsualXStorageV0 storage $ = _usualXStorageV0();
        YieldDataStorage storage yieldStorage = _getYieldDataStorage();

        // Check withdrawal limit
        uint256 maxAssets = maxWithdraw(owner);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxWithdraw(owner, assets, maxAssets);
        }
        // Calculate shares needed
335:    shares = previewWithdraw(assets);
336:    uint256 fee = Math.mulDiv(assets, $.withdrawFeeBps, BASIS_POINT_BASE,
    Math.Rounding.Ceil);

        // Perform withdrawal (exact assets to receiver)
        super._withdraw(_msgSender(), receiver, owner, assets, shares);

        // take the fee
342:    yieldStorage.totalDeposits -= fee;
    }
```

The UsualX.previewWithdraw() function on L335 is following.

```
    function previewWithdraw(uint256 assets) public view override returns (uint256 shares) {
        UsualXStorageV0 storage $ = _usualXStorageV0();
        // Calculate the fee based on the equivalent assets of these shares
396:    uint256 fee = Math.mulDiv(
            assets, $.withdrawFeeBps, BASIS_POINT_BASE - $.withdrawFeeBps, Math.Rounding.Ceil
        );
        // Calculate total assets needed, including fee
        uint256 assetsWithFee = assets + fee;

        // Convert the total assets (including fee) to shares
        shares = _convertToShares(assetsWithFee, Math.Rounding.Ceil);
    }
```

As can be seen, The calculations of withdrawal fee is different in L336 and L396. Since assets refers to the asset amount without fee in the withdraw() and previewWithdraw() functions, the formula of L336 is wrong and the fee of L336 will be smaller than the fee of L396 when withdrawFeeBps>0. As a result, the totalDeposits in L342 will becomes larger than it should be.

## Internal pre-conditions

withdrawFeeBps is larger than zero.

## External pre-conditions

*No response*

## Attack Path

1. Assume that there are multiple users in the UsualX vault.
2. A user withdraws some assets from the vault.
3. totalDeposits becomes larger than it should be. Therefore, the assets of other users will be inflated.

## Impact

Loss of protocol's fee. Broken core functionality because later withdrawers can make a profit.

## PoC

Add the following code to `UsualXUnit.t.sol`.

```
function test_withdrawFee() public {
    // update withdrawFee as 5%
    uint256 fee = 500;
    vm.prank(admin);
    registryAccess.grantRole(WITHDRAW_FEE_UPDATER_ROLE, address(this));
    usualX.updateWithdrawFee(fee);

    // initialize the vault with user1 and user2
    Init memory init;
    init.share[1] = 1e18; init.share[2] = 1e18;
    init.asset[1] = 1e18; init.asset[2] = 1e18;
    init = clamp(init);
    setUpVault(init);
    address user1 = init.user[1];
    address user2 = init.user[2];

    uint256 assetsOfUser2Before = _max_withdraw(user2);

    // user1 withdraw 0.5 ether from the vault
    vm.prank(user1);
    vault_withdraw(0.5e18, user1, user1);

    uint256 assetsOfUser2After = _max_withdraw(user2);

    // compare assets of user2 before withdrawal
    emit log_named_uint("before", assetsOfUser2Before);
    emit log_named_uint("after ", assetsOfUser2After);
}
```

The output log of the above code is the following.

```
Ran 1 test for test/vaults/UsualXUnit.t.sol:UsualXUnitTest
[PASS] test_withdrawFee() (gas: 456410)
Logs:
  before: 1000000000000000100
  after : 1000892857142857243

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 13.10ms (3.33ms CPU
 ↪  time)
```

```
Ran 1 test suite in 20.07ms (13.10ms CPU time): 1 tests passed, 0 failed, 0 skipped
↪  (1 total tests)
```

As can be seen, after user1 withdraws, the assets of user2 becomes inflated.

## Mitigation

Modify `UsualX.withdraw()` function similar to the `UsualX.previewWithdraw()` function as follows.

```
    function withdraw(uint256 assets, address receiver, address owner)
        public
        override
        whenNotPaused
        nonReentrant
        returns (uint256 shares)
    {
        UsualXStorageV0 storage $ = _usualXStorageV0();
        YieldDataStorage storage yieldStorage = _getYieldDataStorage();

        // Check withdrawal limit
        uint256 maxAssets = maxWithdraw(owner);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxWithdraw(owner, assets, maxAssets);
        }
        // Calculate shares needed
        shares = previewWithdraw(assets);
--      uint256 fee = Math.mulDiv(assets, $.withdrawFeeBps, BASIS_POINT_BASE,
↪   Math.Rounding.Ceil);
++      uint256 fee = Math.mulDiv(assets, $.withdrawFeeBps, BASIS_POINT_BASE -
↪   $.withdrawFeeBps, Math.Rounding.Ceil);

        // Perform withdrawal (exact assets to receiver)
        super._withdraw(_msgSender(), receiver, owner, assets, shares);

        // take the fee
        yieldStorage.totalDeposits -= fee;
    }
```

## Discussion

### sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/usual-dao/pegasus/pull/1760

# Disclaimers

Sherlock 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.