SHERLOCK

Security Review For SYMMIO

Public Audit Contest Prepared For: **SYMMIO**
Lead Security Expert: **0x73696d616f**
Date Audited: **March 7 - March 10, 2025**
Final Commit: **cfe1920**

Introduction

Symmio is a permissionless OTC derivatives protocol and clearing layer. This contest focuses on peripheral contracts for our governance token.

Scope

Repository: SYMM-IO/token

Audited Commit: 1d014156b1d9f0ab3259026127b9220eb2da3292

Final Commit: cfe192090c339cffb07d2a50f6ba646299fbcfe0

Files:

- contracts/staking/SymmStaking.sol
- contracts/vesting/SymmVesting.sol
- contracts/vesting/Vesting.sol
- contracts/vesting/interfaces/IMintableERC20.sol
- contracts/vesting/interfaces/IPermit2.sol
- contracts/vesting/interfaces/IPool.sol
- contracts/vesting/interfaces/IRouter.sol
- contracts/vesting/libraries/LibVestingPlan.sol

Final Commit Hash

cfe192090c339cffb07d2a50f6ba646299fbcfe0

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
1	5

Issues Not Fixed and Not Acknowledged

High	Medium
0	0

Security experts who found valid issues

0day
0x23r0
0x73696d616f
0xAristos
0xBecket
0xCNX
0xDarko
0xDemon
0xc0ffEE
0xgremlincat555
0xhammadghazi
0xkmg
0xlucky
0xmechanic
0xpiken
A_Failures_True_Power
Abhan1041
Afriaudit
Akhuemokhan.ETH
Anirruth
Arav
Artur
Audinarey
BAdal-Sharma-09
Beejay
Boy2000
Breeje
BusinessShotgun
CL001
ChaosSR
Cybrid
DenTonylifer
DharkArtz
Drynooo
Edoscoba
ElmInNyc99
EmanHerawy
Flare
Fortis_Audits
Frontrunner
Greed
Hackoor
HaidutiSec
KlosMitSoss
Kyosi
LSH.F.GJ
Limbooo
LonWof-Demon
MSK
Matin
MysteryAuditor
OpaBatyo
Opeyemi
Pablo
Pelz
Pro_King
Ragnarok
Ryonen
SUPERMAN_I4G
SarveshLimaye
Schnilch
Silvermist
SlayerSecurity
The_Rezolvers
Uddercover
Victor_TheOracle
Waydou
X0sauce
X12
Yaneca_b
ZoA
anchabadze
arman
aslanbek
aswinraj94
auditism
auditmasterchef
bladeee
brgltd
buggsy
copperscrewer
dimah7
dobrevaleri
duckee032

durov
edger
eta
farismaulana
future2_22
ge6a
ggbond
gkrastenov
godwinudo
hildingr
ihtishamsudo
ilyadruzh
jo13
justAWanderKid
komane007
korok
krot-0025
lls
moray5554
n1ikh1l
newspacexyz
novaman33
octopus_testjjj
omega
onthehunt
oot2k
osuolale
oxwhite
phoenixv110
redbeans
redtrama
roccomania
santiellena
shui
silver_eth
slavina
spdream
stuart_the_minion
t.aksoy
t0x1c
turvec
udo
vladi319
wickie
x0lohaclohell
y4y
yaioxy
ydlee
zraxx

Issue H-1: USDC rewards will not be distributed if `_updateRewardsStates` is triggered too often

Source: https://github.com/sherlock-audit/2025-03-symm-io-stacking-judging/issues/575

Found by

0xBecket, 0xCNX, A_Failures_True_Power, Artur, Audinarey, Breeje, CL001, Fortis_Audits, Kyosi, Pablo, SlayerSecurity, The_Rezolvers, Victor_TheOracle, X12, aslanbek, bladeee, durov, farismaulana, godwinudo, lls, newspacexyz, onthehunt, stuart_the_minion, wickie, zraxx

Summary

https://github.com/sherlock-audit/2025-03-symm-io-stacking/blob/main/token/contracts/staking/SymmStaking.sol#L402-L423

_updateRewardsStates can be triggered as often as each block (2 seconds) via deposit/withdraw/claim/notifyRewardAmount

e.g. if there's 1209.6e6 USDC rewards for one week (604800 seconds)

https://github.com/sherlock-audit/2025-03-symm-io-stacking/blob/main/token/contracts/staking/SymmStaking.sol#L374

rate = 1209_600000 / 604800 = 2000 ”usdc units” per second

https://github.com/sherlock-audit/2025-03-symm-io-stacking/blob/main/token/contracts/staking/SymmStaking.sol#L194-L202

if SYMM total staked supply is 1_000_000e18 (~26560 usd), and we call `deposit` each block, then perTokenStored will be increased by:

2 * 2000 * 1e18 / 1_000_000e18 = 4_000 / 1_000_000 = 0

Therefore, `perTokenStored` will not increase, but lastUpdated will be increased, therefore users will not receive any USDC rewards for staking.

In this particular example, triggering `_updateRewardsStates` once in 249 blocks would be sufficient, as it would still result in rewards rounding down to zero.

Root Cause

Lack of upscaling for tokens with less than 18 decimals for reward calculations.

Attack Path

1. Attacker calls deposit/withdraw/notifyRewardAmount with any non-zero amount every block (or less often as long as the calculation will still round down to zero)

Impact

High: stakers do not receive rewards in tokens with low decimals (e.g. USDC, USDT).

PoC

1. SYMM total staked supply = 1_000_000e18
2. `notifyRewardAmount` is called with 1209.6 USDC
3. griefer calls `deposit/withdraw` 1 wei of SYMM each 249 blocks for 1 week
4. USDC rewards are stuck in the contract, instead of being distributed to stakers (but can be rescued by admin)

Mitigation

Introduce 1e12 multiplier for reward calculation, and divide the accumulated rewards by 1e12 when they are being claimed.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/SYMM-IO/token/pull/6

Issue M-1: Incorrect initializer modifier in Vesting contract prevents proper initialization

Source: https://github.com/sherlock-audit/2025-03-symm-io-stacking-judging/issues/86

Found by

0xBecket, 0xDemon, ChaosSR, Drynooo, Greed, Hackoor, LonWof-Demon, Ragnarok, The_Rezolvers, Uddercover, ZoA, anchabadze, durov, edger, justAWanderKid, n1ikh1l, octopus_testjjj, t0x1c

Description:

In the Symmio protocol, the `Vesting` contract is designed to be inherited by `SymmVesting`. However, the `__vesting_init()` function in `Vesting` uses the `initializer` modifier instead of the `onlyInitializing` modifier:

https://github.com/sherlock-audit/2025-03-symm-io-stacking/blob/main/token/contracts/vesting/Vesting.sol#L76

```
// Vesting.sol
function __vesting_init(address admin, uint256 _lockedClaimPenalty, address
↪  _lockedClaimPenaltyReceiver)
    public
    initializer
{
    __AccessControlEnumerable_init();
    __Pausable_init();
    __ReentrancyGuard_init();
    // ...rest of initialization...
}
```

Meanwhile, in the inheriting contract: https://github.com/sherlock-audit/2025-03-symm-io-stacking/blob/main/token/contracts/vesting/SymmVesting.sol#L55

```
// SymmVesting.sol
function initialize(
    address admin,
    address _lockedClaimPenaltyReceiver,
    address _pool,
    // ...other parameters...
) public initializer {
    // ...checks...
    __vesting_init(admin, 500000000000000000, _lockedClaimPenaltyReceiver);
```

```
    // ...additional initialization...
}
```

According to OpenZeppelin's documentation and best practices, the `initializer` modifier should only be used in the final initialization function of an inheritance chain, while initialization functions of parent contracts should use the onlyInitializing modifier. This ensures proper initialization when using inheritance. When both parent and child contracts use the initializer modifier, only one of them can actually complete initialization, as the modifier sets a flag that prevents any subsequent calls to functions with the initializer modifier.

Impact:

The vulnerability causes a significant operational issue, preventing inheriting contracts from completing initialization. This could lead to a failure in the deployment of critical protocol components, affecting the overall system functionality.

Recommended Mitigation:

Change the initializer modifier to onlyInitializing in the parent contract:

```
// In Vesting.sol
function __vesting_init(address admin, uint256 _lockedClaimPenalty, address
 ↪  _lockedClaimPenaltyReceiver)
    public
-   initializer
+   onlyInitializing
{
    __AccessControlEnumerable_init();
    __Pausable_init();
    __ReentrancyGuard_init();
    // ...rest of initialization...
}
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/SYMM-IO/token/pull/2/files

Issue M-2: Readding the reward token causes user-RewardPerTokenPaid to be incorrect for some users, resulting in them receiving too many rewards.

Source: https://github.com/sherlock-audit/2025-03-symm-io-stacking-judging/issues/124

Found by

0x23r0, Schnilch, silver_eth

Summary

New users who deposit during the time when the reward token is not added do not get their `userRewardPerTokenPaid` updated for this token, so it remains 0. When the token is re-added, however, `perTokenStored` for this token is not 0 because it retains the previous state. This leads to a situation where users who joined in the meantime when the reward token was not added, can receive all the previous rewards of the token when new rewards are notified, effectively taking them away from other users.

Root Cause

https://github.com/sherlock-audit/2025-03-symm-io-stacking/blob/main/token/contracts/staking/SymmStaking.sol#L319-L328 Here you can see that when removing a reward token, the token is only removed from the `rewardTokens` list without resetting the other state. That means if the token is added again, it takes over the previous state. The problem is that if `perTokenStored` for the reward token is not 0 when it is removed, it will also not be 0 when the token is re-added. If new users make a deposit while the token is not added, they do not get `userRewardPerTokenPaid` updated for this token because the token is no longer in the `rewardTokens` list. Normally `userRewardPerTokenPaid` is always updated before a deposit through `_updateRewardsState` to ensure that a user does not receive rewards that existed before the deposit for the deposited amount: https://github.com/sherlock-audit/2025-03-symm-io-stacking/blob/main/token/contracts/staking/SymmStaking.sol#L406-L418

Internal Pre-conditions

1. There must be a token that is re-added by an authorized address
2. There must be users who start staking during the time when the token is removed and has not yet been re-added

External Pre-conditions

None

Attack Path

1. A new reward token is added
2. User1 deposits
3. Rewards for the token are notified
4. One week passes, and User1 claims his rewards
5. The reward token is removed
6. User2 deposits
7. The reward token is added again
8. Rewards for the token are notified
9. One week passes, and User2 claims his rewards, but he received too many because he also received rewards from the time when the token was first added
10. User2 can no longer claim because there are not enough rewards left in the contract.

Impact

It is very likely that the staking contract will no longer function properly if a reward token is re-added, as some users would receive too many rewards, while others would no longer be able to claim anything due to the lack of rewards. For the users who have too few rewards available, they will also not be able to claim any other reward tokens, as the entire claimRewards function would be reverted.

PoC

The POC can be added to the file `token/tests/symmStaking.behavior.ts` and run with `npx hardhat test --grep "readding token"`:

```
it("readding token", async () => {
    //Reward token is added for the first time
    await symmStaking.connect(admin).configureRewardToken(await
↪   usdcToken.getAddress(), true)

    //User1 stakes 100 SYMM
    await stakingToken.connect(user1).approve(await symmStaking.getAddress(),
↪   e("100"))
    await symmStaking.connect(user1).deposit(e("100"), user1.address)
```

```
    //604.8 USDC are notified as rewards
    await usdcToken.approve(await symmStaking.getAddress(), 604800*1000)
    await symmStaking.notifyRewardAmount([await usdcToken.getAddress()],
↪   [604800*1000])

    time.increaseTo(await time.latest() + 2*30*24*60*60) //Wait 2 months

    await symmStaking.connect(user1).claimRewards() //User1 claims his rewards

    await symmStaking.connect(admin).configureRewardToken(await
↪   usdcToken.getAddress(), false) //The reward token gets removed

    time.increaseTo(await time.latest() + 24*60*60) //Wait 1 day

    //User2 stakes 100 SYMM
    await stakingToken.connect(user2).approve(await symmStaking.getAddress(),
↪   e("100"))
    await symmStaking.connect(user2).deposit(e("100"), user2.address)

    time.increaseTo(await time.latest() + 24*60*60) //Wait 3 months

    await symmStaking.connect(admin).configureRewardToken(await
↪   usdcToken.getAddress(), true) //Reward token is added for the second time

    //1209.6 USDC are notified as rewards
    await usdcToken.approve(await symmStaking.getAddress(), 604800*1000*2)
    await symmStaking.notifyRewardAmount([await usdcToken.getAddress()],
↪   [604800*1000*2])

    time.increaseTo(await time.latest() + 2*7*24*60*60) //Wait 2 weeks

    //Shows that user2 gets all pending rewards and there is nothing left for user1
    console.log("symmStaking pendingRewards before: ", await
↪   symmStaking.pendingRewards(await usdcToken.getAddress()))
    await symmStaking.connect(user2).claimRewards()
    console.log("symmStaking pendingRewards after: ", await
↪   symmStaking.pendingRewards(await usdcToken.getAddress()))
})
```

Mitigation

No response

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/SYMM-IO/token/pull/5

Issue M-3: Bad check in `Vesting.sol::_resetVestingPlans` will prevent users from adding additional liquidity in `SymmVesting.sol`

Source: https://github.com/sherlock-audit/2025-03-symm-io-stacking-judging/issues/509

Found by

0x73696d616f, 0xBecket, 0xgremlincat555, 0xpiken, Afriaudit, Arav, Beejay, BusinessShotgun, Cybrid, Drynooo, OpaBatyo, Ragnarok, Ryonen, Uddercover, X0sauce, aslanbek, copperscrewer, duckee032, farismaulana, future2_22, hildingr, moray5554, onthehunt, oot2k, silver_eth, slavina, t0x1c, zraxx

Summary

The `_resetVestingPlans` check makes it impossible to increase a user's locked tokens if the increase does not push the new amount above the total unlocked tokens. This is problematic as it will prevent users from adding additional liquidity to the `SymmVesting.sol` after a certain number of their lp tokens have been unlocked.

Root Cause

In Vesting.sol:231, the check will cause a revert when a user tries to add additional liquidity

Internal Pre-conditions

The user must already have some vested lp tokens

External Pre-conditions

NIL

Attack Path

NIL

Impact

Users are unable to add additional liquidity

PoC

Follow the guide here to integrate foundry into this codebase. Then add the following test into a new file:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.18;

import {SymmStaking} from "../../contracts/staking/SymmStaking.sol";
import {Symmio} from "../../contracts/token/symm.sol";
import {MockERC20} from "../../contracts/mock/MockERC20.sol";
import {TransparentUpgradeableProxy} from
↪  "@openzeppelin/contracts/proxy/transparent/TransparentUpgradeableProxy.sol";
import {SymmVesting} from "../../contracts/vesting/SymmVesting.sol";
import {Test, console} from "forge-std/Test.sol";

contract TestSuite is Test {
    SymmStaking symmStaking;
    Symmio symm;
    SymmStaking implementation;
    SymmVesting symmVesting;
    SymmVesting vestingImplementation;
    address rewardToken;
    address admin;
    address lockedClaimPenaltyReceiver;
    address pool;
    address router;
    address permit2;
    address vault;
    address usdc;
    address symm_lp;

    function setUp() public {
        admin = makeAddr("admin");
        lockedClaimPenaltyReceiver = makeAddr("lockedClaimPenaltyReceiver");
        pool = 0x94Bf449AB92be226109f2Ed3CE2b297Db94bD995;
        router = 0x76578ecf9a141296Ec657847fb45B0585bCDa3a6;
        permit2 = 0x000000000022D473030F116dDEE9F6B43aC78BA3;
        vault = 0xbA1333333333a1BA1108E8412f11850A5C319bA9;
        usdc = 0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913;
        symm_lp = 0x94Bf449AB92be226109f2Ed3CE2b297Db94bD995;
        symm = Symmio(0x800822d361335b4d5F352Dac293cA4128b5B605f);
        implementation = new SymmStaking();
        vestingImplementation = new SymmVesting();

        TransparentUpgradeableProxy proxy = new
↪  TransparentUpgradeableProxy(address(implementation), admin, "");
        TransparentUpgradeableProxy vestingProxy =
            new TransparentUpgradeableProxy(address(vestingImplementation), admin,
↪  "");
```

```
        symmStaking = SymmStaking(address(proxy));
        symmVesting = SymmVesting(address(vestingProxy));

        vm.startPrank(admin);
        symmStaking.initialize(admin, address(symm));
        symmVesting.initialize(
            admin, lockedClaimPenaltyReceiver, pool, router, permit2, vault,
↪   address(symm), usdc, symm_lp
        );
        rewardToken = address(new MockERC20("Token", "TOK"));
        vm.stopPrank();
    }

    function
↪   testUsersWillBeUnableToProvideLiquidityAfterACertainNumberOfUnlockedTokens()
↪   public {
        //admin creates user vest with symm
        address user = makeAddr("user");
        uint256 userVestAmount = 10e18;
        uint256 totalVestedSymmAmount = 100e18;
        uint256 startTime = block.timestamp;
        uint256 endTime = block.timestamp + 10 days;
        deal(usdc, user, 1000e18);

        address[] memory users = new address[](1);
        users[0] = user;
        uint256[] memory amounts = new uint256[](1);
        amounts[0] = userVestAmount;

        vm.startPrank(admin);
        deal(address(symm), address(symmVesting), totalVestedSymmAmount);
        symmVesting.setupVestingPlans(address(symm), startTime, endTime, users,
↪   amounts);
        vm.stopPrank();

        //user adds half their vested tokens as liquidity
        vm.startPrank(user);
        MockERC20(usdc).approve(address(symmVesting), type(uint256).max);
        symmVesting.addLiquidity(userVestAmount / 2, 0, 0);
        vm.stopPrank();

        //move time so more than half of created symm_lp vesting tokens are unlocked
        vm.warp(block.timestamp + 7 days);

        uint256 secondLiquidityAmount = symmVesting.getLockedAmountsForToken(user,
↪   address(symm));

        //second addLiquidity call will revert with "AlreadyClaimedMoreThanThis"
↪   error
```

```
        vm.startPrank(user);
        MockERC20(usdc).approve(address(symmVesting), type(uint256).max);
        vm.expectRevert();
        symmVesting.addLiquidity(secondLiquidityAmount, 0, 0);
        vm.stopPrank();
    }
}
```

Mitigation

Remove the check:

```
function _resetVestingPlans(address token, address[] memory users, uint256[] memory
↪  amounts) internal {
        if (users.length != amounts.length) revert MismatchArrays();
        uint256 len = users.length;
        for (uint256 i = 0; i < len; i++) {
            address user = users[i];
            uint256 amount = amounts[i];
            _claimUnlockedToken(token, user);
            VestingPlan storage vestingPlan = vestingPlans[token][user];
-           if (amount < vestingPlan.unlockedAmount()) revert
↪  AlreadyClaimedMoreThanThis();
            uint256 oldTotal = vestingPlan.lockedAmount();
            vestingPlan.resetAmount(amount);
            totalVested[token] = totalVested[token] - oldTotal + amount;
            emit VestingPlanReset(token, user, amount);
        }
    }
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/SYMM-IO/token/pull/3

Issue M-4: Malicious User can dilute staking Rewards to a longer timeframe

Source: https://github.com/sherlock-audit/2025-03-symm-io-stacking-judging/issues/595

Found by

0day, 0xAristos, 0xBecket, 0xDarko, 0xc0ffEE, 0xhammadghazi, 0xkmg, 0xlucky, 0xmechanic, 0xpiken, Abhan1041, Akhuemokhan.ETH, Anirruth, Arav, Artur, Audinarey, BAdal-Sharma-09, Boy2000, Breeje, BusinessShotgun, DenTonylifer, DharkArtz, Drynooo, Edoscoba, ElmInNyc99, EmanHerawy, Flare, Fortis_Audits, Frontrunner, HaidutiSec, KlosMitSoss, LSH.F.GJ, Limbooo, LonWof-Demon, MSK, Matin, MysteryAuditor, OpaBatyo, Opeyemi, Pablo, Pelz, Pro_King, Ryonen, SUPERMAN_I4G, SarveshLimaye, Silvermist, SlayerSecurity, Waydou, X0sauce, X12, Yaneca_b, ZoA, arman, aslanbek, aswinraj94, auditism, auditmasterchef, brgltd, buggsy, copperscrewer, dimah7, dobrevaleri, durov, eta, farismaulana, future2_22, ggbond, gkrastenov, hildingr, ihtishamsudo, ilyadruzh, jo13, komane007, korok, krot-0025, moray5554, newspacexyz, novaman33, omega, onthehunt, osuolale, oxwhite, phoenixv110, redbeans, redtrama, roccomania, santiellena, shui, silver_eth, spdream, stuart_the_minion, t.aksoy, t0x1c, turvec, udo, vladi319, x0lohaclohell, y4y, yaioxy, ydlee

Summary

The `SymmStaking` contract allows anyone to add new rewards using the `notifyRewardAmount` function. However, if new rewards are continuously added while existing rewards are still active, the total rewards get spread over a longer period. A malicious actor can exploit this by repeatedly adding tiny amounts, effectively delaying stakers from receiving their full rewards.

Root Cause

- `notifyRewardAmount` function can be called by anyone, with any reward amount.
- Each time it's called, the reward rate is recalculated as:
  - `amount / state.duration` (if the previous reward period has ended).
  - (amount + leftover) / state.duration (if the previous reward period is still ongoing).

The issue arises when an attacker keeps adding tiny amounts (e.g., 1 wei) repeatedly. While the total rewards (amount + leftover) barely change, the duration (state.duration) remains fixed at 1 week, causing the reward rate to drop significantly over time.

Example:

1. Alice is the only staker, and 100 USDC is added as a reward.
2. Halfway through, Alice has earned 50 USDC.
3. A malicious user then adds just 1 wei USDC as a new reward.
4. This recalculates the reward rate, cutting it in half:
    - From `100e6 / 1 week` → `50e6 / 1 week`.
5. The attacker can repeat this process multiple times, continuously lowering the rate.

This DoS-like attack prevents stakers from claiming their rewards in a reasonable timeframe.

Internal Pre-conditions

None.

External Pre-conditions

None.

Attack Path

1. Users stakes in `SymmStaking`.
2. A new reward is notified via `notifyRewardAmount` for the stakers.
3. A malicious user calls `notifyRewardAmount` multiple times with dust values to dilute the reward rate.
4. User get rewards slower than what they were supposed to get.

Impact

Time to gain the intended reward can be arbitrarily increased by malicious users.

PoC

No response

Mitigation

Consider adding restrictions on who can add new reward. Alternatively, implement a minimum amount of reward tokens that can be added to ensure that the total reward amount is meaningfully increased.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/SYMM-IO/token/pull/4

Issue M-5: Double spending attack in the Vesting contract

Source: https://github.com/sherlock-audit/2025-03-symm-io-stacking-judging/issues/650

The protocol has acknowledged this issue.

Found by

0x73696d616f, ge6a

Summary

The function resetVestingPlans() is called by an administrator account and resets vesting plans for a list of users, with the corresponding amount provided as input. The function calls _resetVestingPlans(), where it checks whether the given amount is greater than or equal to the claimed amount for the user. After that, it calls resetAmount() from LibVestingPlan. In this function, the state is updated, the new amount is recorded, and claimedAmount is set to 0.

Root Cause

The issue here is that this can lead to double spending. Even though the user executing the request is trusted, they cannot know whether another transaction has been executed before theirs, in which the user whose vesting plan is being reset has withdrawn their locked amount by paying a penalty fee. If this happens, the user will be able to claim the same amount again after the reset, which would harm other users who might not be able to claim their rewards.

https://github.com/sherlock-audit/2025-03-symm-io-stacking/blob/main/token/contracts/vesting/Vesting.sol#L222-L237

Internal Pre-conditions

None

External Pre-conditions

None

Attack Path

1. Trusted user sends a transaction for executes resetVestingPlans()
2. Regular user subject of this reset sends a transaction that is executed before the first one and claim their locked tokens as they pay a penalty
3. After the reset the user is able to claim the tokens up to amount again

Impact

Loss of funds for the protocol and for the users

PoC

No response

Mitigation

No response

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.