xTRIBE contest
Findings & Analysis Report

2022-07-14

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the xTRIBE smart contract system written in Solidity. The audit contest took place between April 21—April 27 2022.

Wardens

47 Wardens contributed reports to the xTRIBE contest:

1. IIIIIII
2. gzeon
3. VAD37
4. WatchPug (jtp and ming)
5. smiling_heretic
6. hyh
7. rayn
8. 0x52
9. cccz
10. joestakey

11. 0xkatana
12. Dravee
13. robee
14. delfin454000
15. sorrynotsorry
16. defsec
17. catchup
18. teryanarmen
19. oyc_109
20. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
21. MaratCerby
22. 0v3rf10w
23. 0xmint
24. fatima_naz
25. csanuragjain
26. samruna
27. kebabsec (okkothejawa and FlameHorizon)
28. Ruhum
29. hake
30. 0xDjango
31. simon135
32. dipp
33. Tomio
34. Scocco
35. 0xNazgul
36. saian
37. joshie
38. nahnah
39. z3s

40. Funen
41. NoamYakov
42. djxploit
43. 0x1f8b
44. Fitraldys
45. rotcivegaf

This contest was judged by 0xean.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 7 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 7 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 27 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 33 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 xTRIBE contest repository, and is composed of 6 smart contracts written in the Solidity programming language and includes 1,770 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website](#).

Medium Risk Findings (7)

[M-01] `xERC4626.sol` Some users may not be able to withdraw until `rewardsCycleEnd` the due to underflow in `beforeWithdraw()`

Submitted by WatchPug

[xERC4626.sol#L65-L68](#)

```
    function beforeWithdraw(uint256 amount, uint256 shares) internal
        super.beforeWithdraw(amount, shares);
        storedTotalAssets -= amount;
    }
```

[xERC4626.sol#L78-L87](#)

```
    function syncRewards() public virtual {
        uint192 lastRewardAmount_ = lastRewardAmount;
        uint32 timestamp = block.timestamp.safeCastTo32();

        if (timestamp < rewardsCycleEnd) revert SyncError();

        uint256 storedTotalAssets_ = storedTotalAssets;
```

```
        uint256 nextRewards = asset.balanceOf(address(this)) - store

        storedTotalAssets = storedTotalAssets_ + lastRewardAmount_;
        ...
```

`storedTotalAssets` is a cached value of total assets which will only include the `unlockedRewards` when the whole cycle ends.

This makes it possible for `storedTotalAssets -= amount` to revert when the withdrawal amount exceeds `storedTotalAssets`, as the withdrawal amount may include part of the `unlockedRewards` in the current cycle.

Proof of Concept

Given:

- rewardsCycleLength = 100 days
- Alice `deposit()` 100 TRIBE tokens;
- The owner transferred 100 TRIBE tokens as rewards and called `syncRewards()`;
- 1 day later, Alice `redeem()` with all shares, the transaction will revert at `xERC4626.beforeWithdraw()`.

Alice's shares worth 101 TRIBE at this moment, but `storedTotalAssets` = 100, making `storedTotalAssets -= amount` reverts due to underflow.

4. Bob `deposit()` 1 TRIBE tokens;
5. Alice `withdraw()` 101 TRIBE tokens, `storedTotalAssets` becomes `0`;
6. Bob can't even withdraw 1 wei of TRIBE token, as `storedTotalAssets` is now `0`.

If there are no new deposits, both Alice and Bob won't be able to withdraw any of their funds until `rewardsCycleEnd`.

Recommended Mitigation Steps

Consider changing to:

```
function beforeWithdraw(uint256 amount, uint256 shares) internal
    super.beforeWithdraw(amount, shares);
    uint256 _storedTotalAssets = storedTotalAssets;
    if (amount >= _storedTotalAssets) {
        uint256 _totalAssets = totalAssets();
        // _totalAssets - _storedTotalAssets == unlockedRewards
        lastRewardAmount -= _totalAssets - _storedTotalAssets;
        lastSync = block.timestamp;
        storedTotalAssets = _totalAssets - amount;
    } else {
        storedTotalAssets = _storedTotalAssets - amount;
    }
}
```

Joeysantoro (xTRIBE) confirmed, but disagreed with High severity, and commented:

> This is a valid issue, although the risk is probably medium as the affected user could simply wait until the end of the cycle, and this would only occur in an extreme complete withdrawal of the contract.

> As a soft mitigation, I would prefer to simply override maxWithdraw to return `storedTotalAssets_`.

Oxean (judge) decreased severity to Medium and commented:

> I agree with the sponsor here. Assets are not directly lost.

> `3 — High: Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).`

> `2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

> So a medium severity seems more appropriate.

[M-02] First xERC4626 deposit exploit can break share calculation

Submitted by VAD37

Solmate `convertToShares` function follow the formula: `assetDepositAmount * totalShareSupply / assetBalanceBeforeDeposit`.

The share price always return 1:1 with asset token. If everything work normally, share price will slowly increase with time to 1:2 or 1:10 as more rewards coming in.

But right after xERC4626 contract creation, during first cycle, any user can deposit 1 share set `totalSupply = 1`. And transfer token to vault to inflate `totalAssets()` before rewards kick in. (Basically, pretend rewards themselves before anyone can deposit in to get much better share price.)

This can inflate base share price as high as 1:1e18 early on, which force all subsequence deposit to use this share price as base.

Impact

New xERC4626 vault share price can be manipulated right after creation. Which give early depositor greater share portion of the vault during the first cycle.

While deposit token also affected by rounding precision (due to exploit above) that always return lesser amount of share for user.

Proof of Concept

Add these code to `xERC4626Test.t.sol` file to test.

```
    function testExploitNormalCase() public {
        token.mint(address(this), 1e24); // funding
        token.approve(address(xToken), type(uint256).max);
        xToken.syncRewards();
        hevm.warp(1); // skip 1 block from sync rewards. to upda
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 1e18  ", xToken.deposit(1e1
        emit log_named_uint("share price   ", xToken.convertToAs
```

```
        emit log_named_uint("deposit 500e18", xToken.deposit(500
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 500e18", xToken.deposit(500
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_string("fast forward 1 hour to new rewards cycl
        hevm.warp(3601); // new cycle
        emit log_named_uint("deposit 2e18  ", xToken.deposit(2e1
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 500e18", xToken.deposit(500
        emit log_named_uint("share price   ", xToken.convertToAs
        // Share price stay the same 1:1. Due to no rewards have
    }

    function testExploitShare() public {
        token.mint(address(this), 1e24); // funding
        token.approve(address(xToken), type(uint256).max);

        // init total supply as 1:1 share with token as one.
        xToken.deposit(1, address(this));

        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_string("transfer 100e18 fake token rewards to i
        // transfer fake rewards token to xToken contract to inf
        token.transfer(address(xToken), 100e18);
        xToken.syncRewards();
        hevm.warp(1); // skip 1 block from sync rewards. to upda

        // totalSupply() still 1. So current share price is ~ 1e
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 1e18  ", xToken.deposit(1e1
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 500e18", xToken.deposit(500
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 500e18", xToken.deposit(500
        emit log_named_uint("share price   ", xToken.convertToAs
        // After new cycle come around. No rewards have been giv
        // But TotalAsset() have been updated to include fake re
        // this push share price even higher than it should be.
        emit log_string("fast forward 1 hour to new rewards cycl
        hevm.warp(3601); // new cycle
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 2e18  ", xToken.deposit(2e1
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 500e18", xToken.deposit(500
        emit log_named_uint("share price   ", xToken.convertToAs
        // xToken.syncRewards();
```

```
        // hevm.warp(7202); // new cycle
        // Test rounding up value of share
        emit log_named_uint("deposit 1.3e17", xToken.deposit(1.3
        emit log_named_uint("deposit 1.9e17", xToken.deposit(1.9
        emit log_named_uint("deposit 2e17  ", xToken.deposit(2e1
        emit log_named_uint("share price   ", xToken.convertToAs
        emit log_named_uint("deposit 2.5e17", xToken.deposit(2.5
        // token too small will be reverted.
        hevm.expectRevert(abi.encodePacked("ZERO_SHARES"));
        xToken.deposit(1e17, address(this));
        emit log_string("deposit token less than share price amo

        emit log_string("fast forward 1 hour to new rewards cycl
        xToken.syncRewards();
        hevm.warp(7610); // new cycle
        emit log_named_uint("share price   ", xToken.convertToAs

        emit log_string("fast forward 1 hour to new rewards cycl
        xToken.syncRewards();
        hevm.warp(7610+3601); // new cycle
        emit log_named_uint("share price   ", xToken.convertToAs
    }
```

Log Result:

```
Running 2 tests for src\test\xERC4626.t.sol:xERC4626Test
[PASS] testExploitNormalCase() (gas: 286966)
Logs:
  share price   : 1
  deposit 1e18  : 1000000000000000000
  share price   : 1
  deposit 500e18: 500000000000000000000
  share price   : 1
  deposit 500e18: 500000000000000000000
  share price   : 1
  fast forward 1 hour to new rewards cycle
  deposit 2e18  : 2000000000000000000
  share price   : 1
  deposit 500e18: 500000000000000000000
  share price   : 1

[PASS] testExploitShare() (gas: 410737)
Logs:
  share price   : 1
```

```
    transfer 100e18 fake token rewards to inflate share price
    share price   : 100000000000000001
    deposit 1e18  : 9
    share price   : 110000000000000000
    deposit 500e18: 4545
    share price   : 110010976948408342
    deposit 500e18: 4545
    share price   : 110010989010989010
    fast forward 1 hour to new rewards cycle
    share price   : 120989010989010989
    deposit 2e18  : 16
    share price   : 120996050899517332
    deposit 500e18: 4132
    share price   : 120999396135265700
    deposit 1.3e17: 1
    deposit 1.9e17: 1
    deposit 2e17  : 1
    share price   : 121011244434382310
    deposit 2.5e17: 2
    deposit token less than share price amount will be reverted du
    fast forward 1 hour to new rewards cycle
    share price   : 121011846374405794
    fast forward 1 hour to new rewards cycle
    share price   : 121011846374405794

Test result: ok. 2 passed; 0 failed; finished in 20.78ms
```

Recommended Mitigation Steps

This exploit is unique to contract similar to ERC4626. It only works if starting supply equal 0 or very small number and rewards cycle is very short. Or everyone withdraws, total share supply become 0.

This can be easily fix by making sure someone always deposited first so `totalSupply` become high enough that this exploit become irrelevant. Unless in unlikely case someone made arbitrage bot watching vault factory contract.
Just force deposit early token during vault construction as last resort.

Joeysantoro (xTRIBE) commented:

> https://github.com/Rari-Capital/solmate/pull/174/files this is a known issue with 4626. xTRIBE would be initialized safely in this case.

Oxean (judge) decreased severity to Medium and commented:

> Known or unknown this is still a valid attack that isn't mitigated for in the current codebase. Given that there are mitigations that could be integrated on chain (like in the uniswap contracts that burn the first dust amount of LP tokens) , and the warden did demonstrate the attack I am going to downgrade this to medium severity as a "leak of value".

> `2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

[M-03] `ERC20Gauges` : The `_incrementGaugeWeight` function does not check the gauge parameter enough, so the user may lose rewards

Submitted by cccz, also found by 0x52

The _incrementGaugeWeight function is used to increase the user's weight on the gauge. However, in the _incrementGaugeWeight function, it is only checked that the gauge parameter is not in _deprecatedGauges, but not checked that the gauge parameter is in _gauges. If the user accidentally uses the wrong gauge parameter, the function will be executed smoothly without any warning, which will cause user loss reward.

```
function _incrementGaugeWeight(
    address user,
    address gauge,
    uint112 weight,
    uint32 cycle
) internal {
    if (_deprecatedGauges.contains(gauge)) revert InvalidGau
    unchecked {
        if (cycle - block.timestamp <= incrementFreezeWindow
    }

    bool added = _userGauges[user].add(gauge); // idempotent
```

```
        if (added && _userGauges[user].length() > maxGauges && !
            revert MaxGaugeError();

        getUserGaugeWeight[user][gauge] += weight;

        _writeGaugeWeight(_getGaugeWeight[gauge], _add, weight,

        emit IncrementGaugeWeight(user, gauge, weight, cycle);
    }
    ...
    function _writeGaugeWeight(
        Weight storage weight,
        function(uint112, uint112) view returns (uint112) op,
        uint112 delta,
        uint32 cycle
    ) private {
        uint112 currentWeight = weight.currentWeight; // @audit
        // If the last cycle of the weight is before the current
        uint112 stored = weight.currentCycle < cycle ? currentWe
        uint112 newWeight = op(currentWeight, delta); // @audit

        weight.storedWeight = stored;
        weight.currentWeight = newWeight;
        weight.currentCycle = cycle;
    }
```

Proof of Concept

[ERC20Gauges.sol#L257]

Recommended Mitigation Steps

```
    function _incrementGaugeWeight(
        address user,
        address gauge,
        uint112 weight,
        uint32 cycle
    ) internal {
-       if (_deprecatedGauges.contains(gauge)) revert InvalidGau
+       if (_deprecatedGauges.contains(gauge) || !_gauges.contai
        unchecked {
            if (cycle - block.timestamp <= incrementFreezeWindow
        }
```

```
            bool added = _userGauges[user].add(gauge); // idempotent
            if (added && _userGauges[user].length() > maxGauges && !
                revert MaxGaugeError();

            getUserGaugeWeight[user][gauge] += weight;

            _writeGaugeWeight(_getGaugeWeight[gauge], _add, weight,

    }
```

Joeysantoro (xTRIBE) disagreed with High severity and commented:

> This is absolutely a valid logic bug. I disagree with the severity, as it would be user error to increment a gauge which was incapable of receiving any weight. Should be medium.

Oxean (judge) decreased severity to Medium and commented:

> This is a tough one to call between medium and high severity. Assets can directly be lost, but putting the wrong address into ANY function call in general is an easy way for a user to lose funds and isn't unique to this protocol. I am going to side with the sponsor and downgrade to medium severity.

[M-04] In `ERC20Gauges`, contribution to total weight is double-counted when `incrementGauge` is called before `addGauge` for a given gauge.

*Submitted by smiling*heretic_

ERC20Gauges.sol#L214
ERC20Gauges.sol#L257
ERC20Gauges.sol#L248
ERC20Gauges.sol#L465-L469

The impact depends really on how gauges are used by other contracts.

The most obvious consequence I can imagine is that some other contract distributes rewards based on `calculateGaugeAllocation`. However, because `_getStoredWeight(_totalWeight, currentCycle)` is now larger than the real total sum of weights, all rewards are smaller than they should be (because of larger denominator `total`).

There can be also (potentially large) leftover amount of rewards that is never distributed because now sum of `calculateGaugeAllocation(gauge, quantity)` over all gauges with constant `quantity` is less than `quantity`. So value might be lost.

Proof of Concept

I added this test (modified `testCalculateGaugeAllocation`) to `ERC20GaugesTest.t.sol` and it passes.

```
    function testExploit() public {
        token.mint(address(this), 100e18);

        token.setMaxGauges(2);
        token.addGauge(gauge1);

        require(token.incrementGauge(gauge1, 1e18) == 1e18);
        require(token.incrementGauge(gauge2, 1e18) == 2e18);

        // gauge added after incrementing...
        token.addGauge(gauge2);

        hevm.warp(3600); // warp 1 hour to store changes
        require(token.calculateGaugeAllocation(gauge1, 150e18) =
        require(token.calculateGaugeAllocation(gauge2, 150e18) =

        // expected value would be 2e18
        require(token.totalWeight() == 3e18);

        require(token.incrementGauge(gauge2, 2e18) == 4e18);

        // ensure updates don't propagate until stored
        require(token.calculateGaugeAllocation(gauge1, 150e18) =
        require(token.calculateGaugeAllocation(gauge2, 150e18) =

        hevm.warp(7200); // warp another hour to store changes a
```

```
        require(token.calculateGaugeAllocation(gauge1, 125e18) =
        require(token.calculateGaugeAllocation(gauge2, 125e18) =

        // expected value would be 4e18
        require(token.totalWeight() == 5e18);
    }
```

As we can see, we can call `token.incrementGauge(gauge2, 1e18)` before `token.addGauge(gauge2)`. This is because **this check** doesn't revert for gauges that were never added in the first place.

First time the total weight is incremented in `_incrementUserAndGlobalWeights` and 2nd time **here**.

If corrupting state like this is adventurous for someone, he can frontrun `token.addGauge` called by the admin with a call to `incrementGauge` which is permissionless.

Tools Used

Foundry

Recommended Mitigation Steps

Use condition `_gauges.contains(gauge) && !_deprecatedGauges.contains(gauge)` to check if a gauge can be incremented instead of just `!_deprecatedGauges.contains(gauge)`. There's a function `isGauge` in the contract that does exactly this.

Joeysantoro (xTRIBE) commented:

> Duplicate of M-03.

0xean (judge) commented:

> I think this is different enough from M-03 to warrant its own issue and stand alone. Happy to discuss further with sponsor if they are adamant it's a duplicate.

thomas-waite (xTRIBE) confirmed and commented:

> This is caused by and is a unique symptom of the same underlying issue as M-03.

[M-05] `FlywheelCore`'s `setFlywheelRewards` can remove access to reward funds from current users

Submitted by hyh, also found by rayn

FlywheelCore.setFlywheelRewards can remove current reward funds from the current users' reach as it doesn't check that newFlywheelRewards' FlywheelCore is this contract.

If it's not, by mistake or with a malicious intent, the users will lose the access to reward funds as this FlywheelCore will not be approved for any fund access to the new flywheelRewards, while all the reward funds be moved there.

Setting severity to medium as on one hand that's system breaking issue (no rewards can be claimed after that, users are rugged reward-wise), on the other hand setFlywheelRewards function is requiresAuth. Also, a room for operational mistake isn't too small here as new flywheelRewards contract can be correctly configured and not malicious in all other regards.

Proof of Concept

FlywheelCore.setFlywheelRewards doesn't check that newFlywheelRewards' FlywheelCore is this FlywheelCore instance:

FlywheelCore.sol#L164-L171

FlywheelCore is immutable within flywheelRewards and its access to the flywheelRewards' funds is set on construction:

BaseFlywheelRewards.sol#L30

This way if new flywheelRewards contract have any different FlywheelCore then current users' access to reward funds will be irrevocably lost as both claiming functionality and next run of setFlywheelRewards will revert, not being able to transfer any funds from flywheelRewards with `rewardToken.safeTransferFrom(address(flywheelRewards), ...)`:

FlywheelCore.sol#L125

FlywheelCore.sol#L168

As FlywheelCore holds user funds accounting via rewardsAccrued mapping, all these accounts became non-operational, as all the unclaimed rewards will be lost for the users.

Recommended Mitigation Steps

Consider adding the require for `address(newFlywheelRewards.flywheel) == address(flywheelRewards.flywheel)` in setFlywheelRewards so that users always retain funds access.

Joeysantoro (xTRIBE) acknowledged and commented:

> Similar to M-06. I think adding this check makes sense.

Oxean (judge) commented:

> This issue seems distinct enough from M-06 to warrant separate issues. Leaving open and not as a duplicate.

[M-06] `FlywheelCore.setBooster()` can be used to steal unclaimed rewards

Submitted by lllllll

A malicious authorized user can steal all unclaimed rewards and break the reward accounting

Even if the authorized user is benevolent the fact that there is a rug vector available may negatively impact the protocol's reputation. Furthermore since this contract is meant to be used by other projects, the trustworthiness of every project cannot be vouched for.

Proof of Concept

By setting a booster that returns zero for all calls to `boostedBalanceOf()` where the `user` address is not under the attacker's control, and returning arbitrary values for those under his/her control, an attacker can choose specific amounts of `rewardToken` to assign to himself/herself. The attacker can then call `claimRewards()` to withdraw the funds. Any amounts that the attacker assigns to himself/herself over the amount that normally would have been assigned, upon claiming, is taken from other users' unclaimed balances, since tokens are custodied by the `flywheelRewards` address rather than per-user accounts.

```
File: flywheel-v2/src/FlywheelCore.sol

182       /// @notice swap out the flywheel booster contract
183       function setBooster(IFlywheelBooster newBooster) exter
184           flywheelBooster = newBooster;
185
186           emit FlywheelBoosterUpdate(address(newBooster));
187       }
```

[FlywheelCore.sol#L182-L187]

```
File: flywheel-v2/src/FlywheelCore.sol

258           uint256 supplierTokens = address(flywheelBooster)
259               ? flywheelBooster.boostedBalanceOf(strategy, u
260               : strategy.balanceOf(user);
261
262           // accumulate rewards by multiplying user tokens b
263           uint256 supplierDelta = (supplierTokens * deltaInd
264           uint256 supplierAccrued = rewardsAccrued[user] + s
265
266           rewardsAccrued[user] = supplierAccrued;
```

[FlywheelCore.sol#L258-L266]

```
File: flywheel-v2/src/FlywheelCore.sol

119       function claimRewards(address user) external {
120           uint256 accrued = rewardsAccrued[user];
121
```

```
122             if (accrued != 0) {
123                 rewardsAccrued[user] = 0;
124
125                 rewardToken.safeTransferFrom(address(flywheelR
```

[FlywheelCore.sol#L119-L125]

Projects also using `BaseFlywheelRewards` or its child contrats, are implicitly approving infinite transfers by the core

```
File: flywheel-v2/src/rewards/BaseFlywheelRewards.sol

25          constructor(FlywheelCore _flywheel) {
26              flywheel = _flywheel;
27              ERC20 _rewardToken = _flywheel.rewardToken();
28              rewardToken = _rewardToken;
29
30              _rewardToken.safeApprove(address(_flywheel), type(
31          }
```

[BaseFlywheelRewards.sol#L25-L31]

The attacker need not keep the booster set this way - he/she can set it, call `accrue()` for his/her specific user, and unset it, all in the same block.

Recommended Mitigation Steps

Make `flywheelRewards` immutable, or only allow it to change if there are no current users.

[Joeysantoro (xTRIBE) commented]:

> This is a similar issue to one which already affects the SushiSwap masterchef. If rewards are decreased without first calling the `accrue` -equivalent on the masterchef, then previous rewards are lost.

> If trust minimization is a desired property (in my opinion it is), then these functions should be behind timelocks.

> If a user can call accrue before the booster is updated, they can lock in past rewards as they are added onto the rewardsAccrued global state var `266 rewardsAccrued[user] = supplierAccrued;`

> I don't really see this as a vulnerability, but will leave it to the C4 judge.

Oxean (judge) commented:

> I do see this as a vulnerability. Essentially, there is a backdoor by which a privileged address can extract value from users. A timelock would be a potential solution to mitigate some of the risk, as well as the mitigation options presented by the warden.

> `2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.`

> This is a hypothetical attack path with external requirements and deserves the medium severity rating.

[M-07] Incorrect accounting of free weight in `_decrementWeightUntilFree`

Submitted by gzeon

In `_decrementWeightUntilFree`, the free weight is calculated by `balanceOf[user] - getUserWeight[user]` plus weight freed from non-deprecated gauges. The non-deprecated criteria is unnecessary and lead to incorrect accounting of free weight.

Proof of Concept

ERC20Gauges.sol#L547-L583

```
function _decrementWeightUntilFree(address user, uint256 wei
    uint256 userFreeWeight = balanceOf[user] - getUserWeight
```

```
        // early return if already free
        if (userFreeWeight >= weight) return;

        uint32 currentCycle = _getGaugeCycleEnd();

        // cache totals for batch updates
        uint112 userFreed;
        uint112 totalFreed;

        // Loop through all user gauges, live and deprecated
        address[] memory gaugeList = _userGauges[user].values();

        // Free gauges until through entire list or under weight
        uint256 size = gaugeList.length;
        for (uint256 i = 0; i < size && (userFreeWeight + totalF
            address gauge = gaugeList[i];
            uint112 userGaugeWeight = getUserGaugeWeight[user][g
            if (userGaugeWeight != 0) {
                // If the gauge is live (not deprecated), includ
                if (!_deprecatedGauges.contains(gauge)) {
                    totalFreed += userGaugeWeight;
                }
                userFreed += userGaugeWeight;
                _decrementGaugeWeight(user, gauge, userGaugeWeig

                unchecked {
                    i++;
                }
            }
        }

        getUserWeight[user] -= userFreed;
        _writeGaugeWeight(_totalWeight, _subtract, totalFreed, c
    }
```

Consider Alice allocated 3 weight to gauge D, gauge A and gauge B equally where gauge D is depricated

1. Alice call _decrementWeightUntilFree(alice, 2)
2. userFreeWeight = 0
3. gauge D is freed, totalFreed = 0, userFreed = 1
4. (userFreeWeight + totalFreed) < weight, continue to free next gauge

5. gauge A is freed, totalFreed = 1, userFreed = 2
6. (userFreeWeight + totalFreed) < weight, continue to free next gauge
7. gauge B is freed, totalFreed = 2, userFreed = 3
8. All gauge is freed

Alternatively, Alice can

1. Alice call _decrementWeightUntilFree(alice, 1)
2. userFreeWeight = balanceOf[alice] - getUserWeight[alice] = 3 - 3 = 0
3. gauge D is freed, totalFreed = 0, userFreed = 1
4. (userFreeWeight + totalFreed) < weight, continue to free next gauge
5. gauge A is freed, totalFreed = 1, userFreed = 2
6. (userFreeWeight + totalFreed) >= weight, break
7. getUserWeight[alice] -= totalFreed
8. Alice call _decrementWeightUntilFree(alice, 2)
9. userFreeWeight = balanceOf[alice] - getUserWeight[alice] = 3 - 1 = 2
10. (userFreeWeight + totalFreed) >= weight, break
11. Only 2 gauge is freed

Recommended Mitigation Steps

No need to treat deprecated gauge separately.

Joeysantoro (xTRIBE) confirmed and commented:

> This appears correct. Would be for a Tribe dev to validate with a test that certain paths could brick create this incorrect accounting.

Low Risk and Non-Critical Issues

For this contest, 27 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **IIIIIII** received the top score from the judge.

The following wardens also submitted reports: joestakey, hyh, robee, rayn, sorrynotsorry, Dravee, MaratCerby, delfin454000, defsec, Ruhum, teryanarmen, hake, gzeon, VAD37, 0xmint, CertoraInc, fatima_naz, 0xDjango, csanuragjain, samruna, catchup, 0v3rf10w, simon135, oyc_109, kebabsec, *and* dipp.

[L-01] Nonce used for multiple purposes

The nonce mapping used for `permit()` calls is the same as the one used for `delegateBySig()`. This should at the very least be documented so signers know that the order of operations between the two functions matters, and so that `multicall()`s can be organized appropriately

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #1

392         require(nonce == nonces[signer]++, "ERC20MultiVotes:
```

ERC20MultiVotes.sol#L392

[L-02] `multicall()`s involving `permit()` and `delegateBySig()` can be DOSed

Attackers monitoring the blockchain for multicalls can front-run by calling `permit()` and `delegateBySig()` before the `multicall()`, causing it to revert. Have separate flavors of the functions where the `multicall()` data is included in the hash

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #1

392         require(nonce == nonces[signer]++, "ERC20MultiVotes:
```

ERC20MultiVotes.sol#L392

[L-03] Misleading comments

```
File: lib/flywheel-v2/src/FlywheelCore.sol   #1
```

```
82      @return the cumulative amount of rewards accrued to user
```

The cumulative amount of rewards accrued to the user since the last claim
FlywheelCore.sol#L82

[L-04] `require()` should be used instead of `assert()`

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #1

196             assert(queuedRewards.storedCycle == 0 || queuedR
```

FlywheelGaugeRewards.sol#L196

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #2

235         assert(queuedRewards.storedCycle >= cycle);
```

FlywheelGaugeRewards.sol#L235

[N-01] `require()` / `revert()` statements should have descriptive reason strings

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #1

114         require(rewardToken.balanceOf(address(this)) - balar
```

FlywheelGaugeRewards.sol#L114

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #2

153             require(rewardToken.balanceOf(address(this)) - b
```

FlywheelGaugeRewards.sol#L153

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #3

154             require(newRewards <= type(uint112).max); // saf
```

FlywheelGaugeRewards.sol#L154

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #4

195             require(queuedRewards.storedCycle < currentCycle
```

FlywheelGaugeRewards.sol#L195

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #5

200             require(nextRewards <= type(uint112).max); // sa
```

FlywheelGaugeRewards.sol#L200

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #6

345             require(_userGauges[user].remove(gauge));
```

ERC20Gauges.sol#L345

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #7

266             require(_delegates[delegator].remove(delegatee))
```

ERC20MultiVotes.sol#L266

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #8
```

```
352                 require(_delegates[user].remove(delegatee));
```

ERC20MultiVotes.sol#L352

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #9

393         require(signer != address(0));
```

ERC20MultiVotes.sol#L393

[N-02] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

```
File: lib/flywheel-v2/src/FlywheelCore.sol   #1

84     function accrue(ERC20 strategy, address user) public retu
```

FlywheelCore.sol#L84

```
File: lib/flywheel-v2/src/FlywheelCore.sol   #2

101     function accrue(
102         ERC20 strategy,
103         address user,
104         address secondUser
105     ) public returns (uint256, uint256) {
```

FlywheelCore.sol#L101-L105

[N-03] Use a more recent version of solidity

Use a solidity version of at least 0.8.4 to get `bytes.concat()` instead of `abi.encodePacked(<bytes>,<bytes>)`
Use a solidity version of at least 0.8.12 to get `string.concat()` instead of `abi.encodePacked(<str>,<str>)`

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #1

4 pragma solidity ^0.8.0;
```

ERC20MultiVotes.sol#L4

[N-04] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated. If the variable is a local cache of another contract's value, consider making the cache variable internal or private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #1

47     uint32 public immutable gaugeCycleLength;
```

seen in lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol
ERC20Gauges.sol#L47

[N-05] Non-library/interface files should use fixed compiler versions, not floating ones

```
File: lib/xTRIBE/src/xTRIBE.sol   #1

4 pragma solidity ^0.8.0;
```

xTRIBE.sol#L4

[N-06] Typos

```
File: lib/flywheel-v2/src/FlywheelCore.sol   #1

17          The Core contract maintaings three important pieces
```

maintaings
FlywheelCore.sol#L17

```
File: lib/flywheel-v2/src/FlywheelCore.sol   #2

262         // accumulate rewards by multiplying user tokens by
```

rewardsPerToken
FlywheelCore.sol#L262

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #3

230     /// @notice thrown when incremending during the freeze w
```

incremending
ERC20Gauges.sol#L230

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #4

143     /// @notice An event thats emitted when an account chang
```

thats
ERC20MultiVotes.sol#L143

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #5

189      * @param delegatee the receivier of votes.
```

receivier

[ERC20MultiVotes.sol#L189](#)

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #6

364    /*///////////////////////////////////////////////////////////
365                              EIP-712 LOGIC
366    ///////////////////////////////////////////////////////////
```

Did you mean EIP-2612?

[ERC20MultiVotes.sol#L364-L366](#)

[N-07] NatSpec is incomplete

```
File: lib/flywheel-v2/src/FlywheelCore.sol   #1

93     /**
94       @notice accrue rewards for a two users on a strategy
95       @param strategy the strategy to accrue a user's rewards
96       @param user the first user to be accrued
97       @param user the second user to be accrued
98       @return the cumulative amount of rewards accrued to the
99       @return the cumulative amount of rewards accrued to the
100    */
101    function accrue(
102        ERC20 strategy,
103        address user,
104        address secondUser
105    ) public returns (uint256, uint256) {
```

Missing: `@param secondUser`

[FlywheelCore.sol#L93-L105](#)

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #2

130      @param num the number of gauges to return
131    */
132    function gauges(uint256 offset, uint256 num) external vi
```

Missing: `@return`

[ERC20Gauges.sol#L130-L132](#)

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #3

176       @param num the number of gauges to return.
177     */
178     function userGauges(
179         address user,
180         uint256 offset,
181         uint256 num
182     ) external view returns (address[] memory values) {
```

Missing: `@return`

[ERC20Gauges.sol#L176-L182](#)

[N-08] Event is missing `indexed` fields

Each `event` should use three `indexed` fields if there are three or more fields

```
File: lib/flywheel-v2/src/FlywheelCore.sol   #1

66     event AccrueRewards(ERC20 indexed strategy, address index
```

[FlywheelCore.sol#L66](#)

```
File: lib/flywheel-v2/src/FlywheelCore.sol   #2

73     event ClaimRewards(address indexed user, uint256 amount);
```

[FlywheelCore.sol#L73](#)

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #3

44     event CycleStart(uint32 indexed cycleStart, uint256 rewar
```

FlywheelGaugeRewards.sol#L44

```
File: lib/flywheel-v2/src/rewards/FlywheelGaugeRewards.sol   #4

47     event QueueRewards(address indexed gauge, uint32 indexed
```

FlywheelGaugeRewards.sol#L47

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #5

234     event IncrementGaugeWeight(address indexed user, address
```

ERC20Gauges.sol#L234

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #6

237     event DecrementGaugeWeight(address indexed user, address
```

ERC20Gauges.sol#L237

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #7

440     event MaxGaugesUpdate(uint256 oldMaxGauges, uint256 newM
```

ERC20Gauges.sol#L440

```
File: lib/flywheel-v2/src/token/ERC20Gauges.sol   #8

443     event CanContractExceedMaxGaugesUpdate(address indexed a
```

ERC20Gauges.sol#L443

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #9
```

```
102     event MaxDelegatesUpdate(uint256 oldMaxDelegates, uint25
```

ERC20MultiVotes.sol#L102

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #10

105     event CanContractExceedMaxDelegatesUpdate(address indexe
```

ERC20MultiVotes.sol#L105

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #11

135     event Delegation(address indexed delegator, address inde
```

ERC20MultiVotes.sol#L135

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #12

138     event Undelegation(address indexed delegator, address ir
```

ERC20MultiVotes.sol#L138

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #13

141     event DelegateVotesChanged(address indexed delegate, uir
```

ERC20MultiVotes.sol#L141

[N-09] Consider addings checks for signature malleability

```
File: lib/flywheel-v2/src/token/ERC20MultiVotes.sol   #1

380         address signer = ecrecover(
381             keccak256(
```

```
382                 abi.encodePacked(
383                     "\x19\x01",
384                     DOMAIN_SEPARATOR(),
385                     keccak256(abi.encode(DELEGATION_TYPEHASH,
386                 )
387             ),
388             v,
389             r,
390             s
391         );
392         require(nonce == nonces[signer]++, "ERC20MultiVotes:
393         require(signer != address(0));
394         _delegate(signer, delegatee);
```

[ERC20MultiVotes.sol#L380-L394]

[0xean (judge) commented]:

> The severities listed in this QA submission are correct as-is.

Gas Optimizations

For this contest, 33 reports were submitted by wardens detailing gas optimizations. The [report highlighted below] by **0xkatana** received the top score from the judge.

The following wardens also submitted reports: [Dravee], [IIIIIII], [delfin454000], [joestakey], [catchup], [Tomio], [defsec], [oyc_109], [robee], [Scocco], [0xNazgul], [0v3rf10w], [saian], [joshie], [CertoraInc], [fatima_naz], [nahnah], [teryanarmen], [z3s], [Funen], [NoamYakov], [kebabsec], [sorrynotsorry], [djxploit], [gzeon], [0x1f8b], [Fitraldys], [rayn], [samruna], [rotcivegaf], [0xmint], *and* [csanuragjain].

[G-01] Redundant zero initialization

Solidity does not recognize null as a value, so uint variables are initialized to zero. Setting a uint variable to zero is redundant and can waste gas.

There are several places where an int is initialized to zero, which looks like:

```
uint256 amount = 0;
```

Instances in code:
ERC20Gauges.sol#L134
ERC20Gauges.sol#L184
ERC20Gauges.sol#L307
ERC20Gauges.sol#L384
ERC20Gauges.sol#L564
ERC20MultiVotes.sol#L346
ERC20MultiVotes.sol#L79
xTRIBE.sol#L95

Recommended Mitigation Steps

Remove the redundant zero initialization

```
uint256 amount;
```

[G-02] Use prefix not postfix in loops

Using a prefix increment (++i) instead of a postfix increment (i++) saves gas for each loop cycle and so can have a big gas impact when the loop executes on a large number of elements.

There are several examples of this:
Multicall.sol#L14
FlywheelGaugeRewards.sol#L189
ERC20MultiVotes.sol#L346
ERC20Gauges.sol#L137
ERC20Gauges.sol#L187
ERC20Gauges.sol#L314
ERC20Gauges.sol#L391
ERC20Gauges.sol#L576
xTRIBE.sol#L99

Recommended Mitigation Steps

Use prefix not postfix to increment in a loop.

[G-03] Short require strings save gas

Strings in solidity are handled in 32 byte chunks. A require string longer than 32 bytes uses more gas. Shortening these strings will save gas.

One cases of this gas optimization was found
34 chars
ERC20MultiVotes.sol#L379

Recommended Mitigation Steps

Shorten all require strings to less than 32 characters.

[G-04] Use != 0 instead of > 0

Using `> 0` uses slightly more gas than using `!= 0`. Use `!= 0` when comparing uint variables to zero, which cannot hold values below zero

Locations where this was found include:
PeripheryPayments.sol#L38
PeripheryPayments.sol#L45
PeripheryPayments.sol#L60
PeripheryPayments.sol#L66
FlywheelCore.sol#L167
FlywheelCore.sol#L218
ERC20Gauges.sol#L467
ERC20Gauges.sol#L487
ERC20MultiVotes.sol#L287

Recommended Mitigation Steps

Replace `> 0` with `!= 0` to save gas.

[G-05] Cache array length before loop

Caching the array length outside a loop saves reading it on each iteration, as long as the array's length is not changed during the loop. This saves gas.

This optimization is already used in some places, but is not used in this place:
Multicall.sol#L14

Recommended Mitigation Steps

Cache the array length before the for loop.

[G-06] Bitshift for divide by 2

When multiply or dividing by a power of two, it is cheaper to bitshift than to use standard math operations.

There is a divide by 2 operation on this line:
ERC20MultiVotes.sol#L94

Recommended Mitigation Steps

Bitshift right by one bit instead of dividing by 2 to save gas.

[G-07] Use simple comparison in trinary logic

The comparison operators >= and <= use more gas than >, <, or ==. Replacing the >= and ≤ operators with a comparison operator that has an opcode in the EVM saves gas.

The existing code is:
FlywheelDynamicRewards.sol#L50

```
uint32 latest = timestamp >= cycle.end ? cycle.end : timestamp;
```

A simple comparison can be used for gas savings by reversing the logic:

```
uint32 latest = timestamp < cycle.end ? timestamp : cycle.end;
```

Recommended Mitigation Steps

Replace the comparison operator and reverse the logic to save gas using the suggestions above.

[G-08] Use simple comparison in if statement

The comparison operators >= and <= use more gas than >, <, or ==. Replacing the >= and ≤ operators with a comparison operator that has an opcode in the EVM saves gas.

The existing code is:
ERC20Gauges.sol#L37-L39

```
if (_incrementFreezeWindow >= _gaugeCycleLength) revert Incremer
gaugeCycleLength = _gaugeCycleLength;
incrementFreezeWindow = _incrementFreezeWindow;
```

A simple comparison can be used for gas savings by reversing the logic:

```
if (_incrementFreezeWindow < _gaugeCycleLength) {
gaugeCycleLength = _gaugeCycleLength;
incrementFreezeWindow = _incrementFreezeWindow;
} else {
revert IncrementFreezeError();
}
```

Recommended Mitigation Steps

Replace the comparison operator and reverse the logic to save gas using the suggestions above.

[G-09] Use calldata instead of memory for function parameters

Use calldata instead of memory for function parameters. Having function arguments use calldata instead of memory can save gas.

There are several cases of function arguments using memory instead of calldata:
ENSReverseRecord.sol#L22
ENSReverseRecord.sol#L26
FlywheelCore.sol#L210

Recommended Mitigation Steps

Change function arguments from memory to calldata.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top