Tribe DAO - Flywheel v2, xTRIBE, xERC4626

Date	April 2022
Auditors	David Oz Kashi, Christian Goll, George Kobakhidze

1 Executive Summary

This report presents the results of our engagement with **Tribe DAO** to review **Flywheel v2, xTRIBE, xERC4626**.

The review was conducted over two weeks, from **April 4, 2022** to **April 15, 2022** by **David Oz Kashi**, **Christian Goll**, and **George Kobakhidze**. A total of 30 person-days were spent.

2 Scope

Our review focused on three different repositories (Flywheel v2, xTRIBE, and xERC4626). Commit hashes are `733d0e1f18090796f07d3f4c208d1ee1f89819c7`, `44d5a278c4c9655f59fdc64b08d8cebb941a2204`, `0a0ab2c7de955cf8aa1a2575c8d7531ac12bf6df` respectively. The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the **Tribe DAO** team, we identified the following priorities for our review:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

3 Recommendations

3.1 Consider adding zero address checks

The Tribe DAO codebase inherits from the solmate library which is designed for maximum gas efficiency, often at the cost of user safety. Tribe DAO makes use of solmate's ERC20 implementation in particular which forgoes zero address checks in it's transfer functions. Consider adding these checks in logic that makes use of ERC20 `transfer` and `transferFrom` to avoid losses that occur due to user error.

3.2 Improve gas optimization

The team has done an amazing job at gas optimization however, we still managed to find a few instances where there could be further improvement. The for-loops implemented in the codebase exclusively make use of a post-increment whereas a pre-increment (++i) is much more efficient, especially in larger for-loops. This is because post-increments store a copy of the old value in memory while pre-increments do not.

3.3 Improve code readability ✓ Fixed

Resolution

Fixed in fei-protocol/ERC4626@`0515d6e` & fei-protocol/flywheel-v2@`1144dd8`

Examples

Some parts are inconsistent with the rest of the codebase despite meaning to do exactly the same thing. In particular, the calculation of cycle boundaries in `xERC4626` doesn't have brackets around the division operation.

code-erc4626/src/xERC4626.sol:L40-L40

```
rewardsCycleEnd = block.timestamp.safeCastTo32() / rewardsCycleLength * rewardsCycleLength;
```

code-erc4626/src/xERC4626.sol:L90

```
uint32 end = (timestamp + rewardsCycleLength) / rewardsCycleLength * rewardsCycleLength;
```

As opposed to these examples:

code-flywheel-v2/src/token/ERC20Gauges.sol:L89-L94

```
function _getGaugeCycleEnd() internal view returns (uint32) {
    uint32 nowPlusOneCycle = block.timestamp.safeCastTo32() + gaugeCycleLength;
    unchecked {
        return (nowPlusOneCycle / gaugeCycleLength) * gaugeCycleLength; // cannot divide by zero and always <= nowPlusOneCycle so
    }
}
```

code-flywheel-v2/src/rewards/FlywheelGaugeRewards.sol:L90

```
gaugeCycle = (block.timestamp.safeCastTo32() / gaugeCycleLength) * gaugeCycleLength;
```

code-flywheel-v2/src/rewards/FlywheelGaugeRewards.sol:L101-L103

```
function queueRewardsForCycle() external requiresAuth returns (uint256 totalQueuedForCycle) {
    // next cycle is always the next even divisor of the cycle length above current block timestamp.
    uint32 currentCycle = (block.timestamp.safeCastTo32() / gaugeCycleLength) * gaugeCycleLength;
```

code-flywheel-v2/src/rewards/FlywheelGaugeRewards.sol:L133-L135

```
function queueRewardsForCyclePaginated(uint256 numRewards) external requiresAuth {
    // next cycle is always the next even divisor of the cycle length above current block timestamp.
    uint32 currentCycle = (block.timestamp.safeCastTo32() / gaugeCycleLength) * gaugeCycleLength;
```

In `ERC20MultiVotes.sol`, there are four different functions with two of each group bearing the same name. Luckily, this isn't a big issue thanks to how function selectors work however, the developers are encouraged to use different names to avoid confusion and improve readability. Consider Renaming the function in lines 211 to `_delegateAndUndelegate` or something more fitting and renaming the function in lines 207 to 209 to `delegateAll/undelegateAndDelegateAll`.

code-flywheel-v2/src/token/ERC20MultiVotes.sol:L211

```
function _delegate(address delegator, address newDelegatee) internal virtual {
```

code-flywheel-v2/src/token/ERC20MultiVotes.sol:L229-L233

```
function _delegate(
    address delegator,
    address delegatee,
    uint256 amount
) internal virtual {
```

code-flywheel-v2/src/token/ERC20MultiVotes.sol:L189-L191

```
function delegate(address delegatee, uint256 amount) public virtual {
    _delegate(msg.sender, delegatee, amount);
}
```

code-flywheel-v2/src/token/ERC20MultiVotes.sol:L207-L209

```
function delegate(address newDelegatee) external virtual {
    _delegate(msg.sender, newDelegatee);
}
```

4 Observations

- `xERC4626.syncRewards()` safecasts `block.timestamp` to uint32 using the `solmate.SafeCastLib` library, which will revert once `block.timestamp > max(uint32)`. This will happen in 84 years on February 7th, 2106.

5 Findings: Flywheel

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major 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.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 Reactivated gauges can't queue up rewards Major ✓ Fixed

Resolution

Fixed in fei-protocol/flywheel-v2@`e765d24` by making it so all gauges are always included in cycles, thus keeping in sync their `storedCycle` values with the contract's state and passing them to the rest of the contract. Downstream `gaugeToken. calculateGaugeAllocation` now handles deprecated gauges by returning 0 for them.

Description

Active gauges as set in `ERC20Gauges.addGauge()` function by authorised users get their rewards queued up in the `FlywheelGaugeRewards._queueRewards()` function. As part of it, their associated struct `QueuedRewards` updates its `storedCycle` value to the cycle in which they get queued up:

code-flywheel-v2/src/rewards/FlywheelGaugeRewards.sol:L202-L206

```
gaugeQueuedRewards[gauge] = QueuedRewards({
    priorCycleRewards: queuedRewards.priorCycleRewards + completedRewards,
    cycleRewards: uint112(nextRewards),
    storedCycle: currentCycle
});
```

However, these gauges may be deactivated in `ERC20Gauges.removeGauge()`, and they will now be ignored in either `FlywheelGaugeRewards.queueRewardsForCycle()` or `FlywheelGaugeRewards.queueRewardsForCyclePaginated()` because both use `gaugeToken.gauges()` to get the set of gauges for which to queue up rewards for the cycle, and that only gives active gauges. Therefore, any updates `FlywheelGaugeRewards` makes to its state will not be done to deactivated gauges' `QueuedRewards` structs. In particular, the `gaugeCycle` contract state variable will keep advancing throughout its cycles, while `QueuedRewards.storedCycle` will retain its previously set value, which is the cycle where it was queued and not 0.

Once reactivated later with at least 1 full cycle being done without it, it will produce issues. It will now be returned by `gaugeToken.gauges()` to be processed in either `FlywheelGaugeRewards.queueRewardsForCycle()` or `FlywheelGaugeRewards.queueRewardsForCyclePaginated()`, but, once the reactivated gauge is passed to `_queueRewards()`, it will fail an assert:

code-flywheel-v2/src/rewards/FlywheelGaugeRewards.sol:L196

```
assert(queuedRewards.storedCycle == 0 || queuedRewards.storedCycle >= lastCycle);
```

This is because it already has a set value from the cycle it was processed in previously (i.e. `storedCycle>0`), and, since that cycle is at least 1 full cycle behind the state contract, it will also not pass the second condition `queuedRewards.storedCycle >= lastCycle`.

The result is that this gauge is locked out of queuing up for rewards because `queuedRewards.storedCycle` is only synchronised with the contract's cycle later in `_queueRewards()` which will now always fail for this gauge.

Recommendation

Account for the reactivated gauges that previously went through the rewards queue process, such as introducing a separate flow for newly activated gauges. However, any changes such as removing the above mentioned `assert()` should be carefully validated for other downstream logic that may use the `QueuedRewards.storedCycle` value. Therefore, it is recommended to review the state transitions as opposed to only passing this specific check.

5.2 Reactivated gauges have incorrect accounting for the last cycle's rewards Medium ✓ Fixed

Resolution

Fixed in fei-protocol/flywheel-v2@`e765d24` by making it so all gauges are always included in cycles, thus keeping in sync their `storedCycle` values with the contract's state.

Description

As described in issue 5.1, reactivated gauges that previously had queued up rewards have a mismatch between their `storedCycle` and contract's `gaugeCycle` state variable.

Due to this mismatch, there is also a resulting issue with the accounting logic for its completed rewards:

code-flywheel-v2/src/rewards/FlywheelGaugeRewards.sol:L198

```
uint112 completedRewards = queuedRewards.storedCycle == lastCycle ? queuedRewards.cycleRewards : 0;
```

Consequently, this then produces an incorrect value for `QueuedRewards.priorCycleRewards`:

code-flywheel-v2/src/rewards/FlywheelGaugeRewards.sol:L203

```
priorCycleRewards: queuedRewards.priorCycleRewards + completedRewards,
```

As now `completedRewards` will be equal to 0 instead of the previous cycle's rewards for that gauge. This may cause a loss of rewards accounted for this gauge as this value is later used in `getAccruedRewards()`.

Recommendation

Consider changing the logic of the check so that `storedCycle` values further in the past than `lastCycle` may produce the right rewards return for this expression, such as using `<=` instead of `==` and adding an explicit check for `storedCycle == 0` to account for the initial scenario.

5.3 Lack of input validation in `delegateBySig` Minor ✓ Fixed

Resolution

Fixed in fei-protocol/flywheel-v2@`e765d24` by reverting for `signer = address(0)`

Description

ERC20MultiVotes.sol makes use of `ecrecover()` in `delegateBySig` to return the address of the message signer. `ecrecover()` typically returns address(0x0) to indicate an error; however, there's no zero address check in the function logic. This might not be exploitable though, as `delegate(0x0, arbitraryAddress)` might always return zero votes (in freeVotes). Additionally, `ecrecover()` can be forced to return a random address by messing with the parameters. Although this is extremely rare and will likely resolve to zero free votes most times, this might return a random address and delegate someone else's votes.

Examples

code-flywheel-v2/src/token/ERC20MultiVotes.sol:L364-L387

```
function delegateBySig(
    address delegatee,
    uint256 nonce,
    uint256 expiry,
    uint8 v,
    bytes32 r,
    bytes32 s
) public {
    require(block.timestamp <= expiry, "ERC20MultiVotes: signature expired");
    address signer = ecrecover(
        keccak256(
            abi.encodePacked(
                "\x19\x01",
                DOMAIN_SEPARATOR(),
                keccak256(abi.encode(DELEGATION_TYPEHASH, delegatee, nonce, expiry))
            )
        ),
        v,
        r,
        s
    );
    require(nonce == nonces[signer]++, "ERC20MultiVotes: invalid nonce");
    _delegate(signer, delegatee);
}
```

Recommendation

Introduce a zero address check i.e `require signer!=address(0)` and check if the recovered signer is an expected address. Refer to ERC20's permit for inspiration.

5.4 Decreasing maxGauges does not account for users' previous gauge list size. Minor ✓ Fixed

Resolution

Fixed in fei-protocol/flywheel-v2@ `e765d24` by documenting.

Description

`ERC20Gauges` contract has a `maxGauges` state variable meant to represent the maximum amount of gauges a user can allocate to. As per the natspec, it is meant to protect against gas DOS attacks upon token transfer to allow complicated transactions to fit in a block. There is also a function `setMaxGauges` for authorised users to decrease or increase this state variable.

code-flywheel-v2/src/token/ERC20Gauges.sol:L499-L504

```
function setMaxGauges(uint256 newMax) external requiresAuth {
    uint256 oldMax = maxGauges;
    maxGauges = newMax;

    emit MaxGaugesUpdate(oldMax, newMax);
}
```

However, if it is decreased and there are users that have already reached the previous maximum that was larger, there may be unexpected behavior. All of these users' gauges will remain active and manageable, such as have user gauge weights incremented or decremented. So it could be possible that for such a user address `user_address`, `numUserGauges(user_address) > maxGauges`. While in the current contract logic this does not cause issues, `maxGauges` is a public variable that may be used by other systems. If unaccounted for, this discrepancy between the contract's `maxGauges` and the users' actual number of gauges given by `numUserGauges()` could, for example, cause gauges to be skipped or fail loops bounded by `maxGauges` in other systems' logic that try and go through all user gauges.

Recommendation

Either document the potential discrepancy between the user gauges size and the `maxGauges` state variable, or limit `maxGauges` to be only called within the contract thereby forcing other contracts to retrieve user gauge list size through `numUserGauges()`.

5.5 Decrementing a gauge by 0 that is not in the user gauge list will fail an assert. Minor ✓ Fixed

Resolution

Fixed in fei-protocol/flywheel-v2@ `e765d24` by implementing auditor's recommendation.

Description

`ERC20Gauges._decrementGaugeWeight` has an edge case scenario where a user can attempt to decrement a `gauge` that is not in the user gauge list by 0 `weight`, which would trigger a failure in an assert.

code-flywheel-v2/src/token/ERC20Gauges.sol:L333-L345

```
function _decrementGaugeWeight(
    address user,
    address gauge,
    uint112 weight,
    uint32 cycle
) internal {
    uint112 oldWeight = getUserGaugeWeight[user][gauge];

    getUserGaugeWeight[user][gauge] = oldWeight - weight;
    if (oldWeight == weight) {
        // If removing all weight, remove gauge from user list.
        assert(_userGauges[user].remove(gauge));
    }
```

As `_decrementGaugeWeight`, `decrementGauge`, or `decrementGauges` don't explicitly check that a `gauge` belongs to the user, the contract logic continues with its operations in `_decrementGaugeWeight` for any gauges passed to it. In general this is fine because if a user tries to decrement non-zero `weight` from a `gauge` they have no allocation to, thus getting `getUserGaugeWeight[user][gauge]=0`, there would be a revert due to a negative value being passed to `getUserGaugeWeight[user][gauge]`

code-flywheel-v2/src/token/ERC20Gauges.sol:L339-L341

```
uint112 oldWeight = getUserGaugeWeight[user][gauge];

getUserGaugeWeight[user][gauge] = oldWeight - weight;
```

However, passing a `weight=0` parameter with a `gauge` that doesn't belong to the user, would successfully process that line. This would then be followed by an evaluation `if (oldWeight == weight)`, which would also succeed since both are 0, to finally reach an assert that will verify a remove of that `gauge` from the user gauge list. However, it will fail since it was never there in the first place.

code-flywheel-v2/src/token/ERC20Gauges.sol:L344

```
assert(_userGauges[user].remove(gauge));
```

Although an edge case with no effect on contract state's health, it may happen with front end bugs or incorrect user transactions, and it is best not to have asserts fail.

Recommendation

Replace `assert()` with a `require()` or verify that the gauge belongs to the user prior to performing any operations.

5.6 Undelegating 0 votes from an address who is not a delegate of a user will fail an assert. Minor

✓ Fixed

Resolution
Fixed in fei-protocol/flywheel-v2@ `e765d24` by implementing auditor's recommendation.

Description

Similar scenario with issue 5.5. `ERC20MultiVotes._undelegate` has an edge case scenario where a user can attempt to undelegate from a `delegatee` that is not in the user delegates list by 0 `amount`, which would trigger a failure in an assert.

code-flywheel-v2/src/token/ERC20MultiVotes.sol:L251-L260

```
function _undelegate(
    address delegator,
    address delegatee,
    uint256 amount
) internal virtual {
    uint256 newDelegates = _delegatesVotesCount[delegator][delegatee] - amount;

    if (newDelegates == 0) {
        assert(_delegates[delegator].remove(delegatee)); // Should never fail.
    }
```

As `_undelegate`, or `undelegate` don't explicitly check that a `delegatee` belongs to the user, the contract logic continues with its operations in `_undelegate` for the `delegatee` passed to it. In general this is fine because if a user tries to undelegate non-zero `amount` from a `delegatee` they have no votes delegated to, thus getting `_delegatesVotesCount[delegator][delegatee]=0`, there would be a revert due to a negative value being passed to `uint256 newDelegates`

code-flywheel-v2/src/token/ERC20MultiVotes.sol:L256

```
uint256 newDelegates = _delegatesVotesCount[delegator][delegatee] - amount;
```

However, passing a `amount=0` parameter with a `delegatee` that doesn't belong to the user, would successfully process that line. This would then be followed by an evaluation `if (newDelegates == 0)`, which would succeed, to finally reach an assert that will verify a remove of that `delegatee` from the user delegates list. However, it will fail since it was never there in the first place.

code-flywheel-v2/src/token/ERC20MultiVotes.sol:L259

```
assert(_delegates[delegator].remove(delegatee)); // Should never fail.
```

Although an edge case with no effect on contract state's health, it may happen with front end bugs or incorrect user transactions, and it is best not to have asserts fail, as per the dev comment in that line "// Should never fail".

Recommendation

Replace `assert()` with a `require()` or verify that the delegatee belongs to the user prior to performing any operations.

6 Findings: xTRIBE

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major 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.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 `xTRIBE.emitVotingBalances` - `DelegateVotesChanged` event can be emitted by anyone

Medium ✓ Fixed

Resolution
Fixed in fei-protocol/xTRIBE@`ea9705b` by adding authentication.

Description

`xTRIBE.emitVotingBalances` is an external function without authentication constraints. It means anyone can call it and emit `DelegateVotesChanged` which may impact other layers of code that rely on these events.

Examples

code-xTRIBE/src/xTRIBE.sol:L89-L99

```
function emitVotingBalances(address[] calldata accounts) external {
    uint256 size = accounts.length;

    for (uint256 i = 0; i < size; ) {
        emit DelegateVotesChanged(accounts[i], 0, getVotes(accounts[i]));

        unchecked {
            i++;
        }
    }
}
```

Recommendation

Consider restricting access to this function for allowed accounts only.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
/src/token/ERC20Gauges.sol	e4511ee6c1bd43aeb634e55f3e634e2e2d026cab
/src/token/ERC20MultiVotes.sol	2c6d5d10cd7138c6e8a29b2099b03179b2f3f48a
/src/rewards/FlywheelGaugeRewards.sol	5f7072a425c0f71f8af3741b9624ee2e59311493
/src/xTRIBE.sol	56ae0fe23413cbcbb50853e37c0cced19a7438eb
/src/xERC4626.sol	8fb3f32849c17bb4b220cbd0db8fe1392575bfa5

Appendix 2 - Disclosure

ConsenSys Diligence ("CD") typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.