Fei Tribechief

Book your 1-Day Security Spot Check

BOOK NOW

Date	July 2021
Auditors	Sergii Kravchenko, Martin Ortner, David Oz Kashi

1 Executive Summary

This report presents the results of our engagement with **Fei Protocol** to review their new staking contracts.

The review was conducted over one week, from **July 12th, 2021** to **July 16, 2021** by **Sergii Kravchenko**, **Martin Ortner** and **David Oz Kashi**. A total of 15 person-days were spent.

2 Scope

Our review focused on the commit hash `3c55d72aaf6f1f60850165a7a9b2e4d59c380551`. The primary focus was to review the new staking component:

- StakingTokenWrapper.sol
- TribalChief.sol

Additionally, we made a superficial review of the Compound PCV contracts.

2.1 Objectives

Together with the **Fei Protocol** 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 Findings

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.

3.1 TribalChief - A wrong `user.rewardDebt` value is calculated during the `withdrawFromDeposit` function call Critical

Description

When withdrawing a single deposit, the reward debt is updated:

contracts/staking/TribalChief.sol:L468-L474

```
uint128 virtualAmountDelta = uint128( ( amount * poolDeposit.multiplier ) / SCALE_FACTOR );

// Effects
poolDeposit.amount -= amount;
user.rewardDebt = user.rewardDebt - toSigned128(user.virtualAmount * pool.accTribePerShare) / toSigned128(ACC_TRIBE_PRECISION);
user.virtualAmount -= virtualAmountDelta;
pool.virtualTotalSupply -= virtualAmountDelta;
```

Instead of the `user.virtualAmount` in reward debt calculation, the `virtualAmountDelta` should be used. Because of that bug, the reward debt is much lower than it would be, which means that the reward itself will be much larger during the harvest. By making multiple deposit-withdraw actions, any user can steal all the Tribe tokens from the contract.

Recommendation

Use the `virtualAmountDelta` instead of the `user.virtualAmount`.

3.2 TribalChief - Setting the `totalAllocPoint` to zero shouldn't be allowed Medium

Description

`TribalChief.updatePool` will revert in the case `totalAllocPoint = 0`, which will essentially cause users' funds and rewards to be locked.

Recommendation

`TribalChief.add` and `TribalChief.set` should assert that `totalAllocPoint > 0`. A similar validation check should be added to `TribalChief.updatePool` as well.

3.3 TribalChief - Unlocking users' funds in a pool where a multiplier has been increased is missing

Medium

Description

When a user deposits funds to a pool, the current multiplier in use for this pool is being stored **locally** for this deposit. The value that is used later in a withdrawal operation is the **local one**, and not the one that is changing when a `governor` calls `governorAddPoolMultiplier`. It means that a decrease in the multiplier value for a given pool does not affect users that already deposited, but an increase does. Users that had already deposited should have the right to withdraw their funds when the multiplier for their pool increases by the `governor`.

Examples

code/contracts/staking/TribalChief.sol:L143-L158

```
function governorAddPoolMultiplier(
    uint256 _pid,
    uint64 lockLength,
    uint64 newRewardsMultiplier
) external onlyGovernor {
    PoolInfo storage pool = poolInfo[_pid];
    uint256 currentMultiplier = rewardMultipliers[_pid][lockLength];
    // if the new multplier is less than the current multiplier,
    // then, you need to unlock the pool to allow users to withdraw
    if (newRewardsMultiplier < currentMultiplier) {
        pool.unlocked = true;
    }
    rewardMultipliers[_pid][lockLength] = newRewardsMultiplier;

    emit LogPoolMultiplier(_pid, lockLength, newRewardsMultiplier);
}
```

Recommendation

Replace the `<` operator with `>` in `TribalChief` line 152.

3.4 TribalChief - Unsafe down-castings Medium

Description

`TribalChief` consists of multiple unsafe down-casting operations. While the usage of types that can be packed into a single storage slot is more gas efficient, it may introduce hidden risks in some cases that can lead to loss of funds.

Examples

Various instances in `TribalChief`, including (but not necessarily only) :

code/contracts/staking/TribalChief.sol:L429

```
user.rewardDebt = int128(user.virtualAmount * pool.accTribePerShare) / toSigned128(ACC_TRIBE_PRECISION);
```

code/contracts/staking/TribalChief.sol:L326

```
pool.accTribePerShare = uint128(pool.accTribePerShare + ((tribeReward * ACC_TRIBE_PRECISION) / virtualSupply));
```

code/contracts/staking/TribalChief.sol:L358

```
userPoolData.rewardDebt += int128(virtualAmountDelta * pool.accTribePerShare) / toSigned128(ACC_TRIBE_PRECISION);
```

Recommendation

Given the time constraints of this audit engagement, we could not verify the implications and provide mitigation actions for each of the unsafe down-castings operations. However, we do recommend to either use numeric types that use 256 bits, or to add proper validation checks and handle these scenarios to avoid silent over/under-flow errors. Keep in mind that reverting these scenarios can sometimes lead to a denial of service, which might be harmful in some cases.

3.5 EthCompoundPCVDeposit - should provide means to recover ETH Medium

Description

`EthCompoundPCVDeposit` accepts `ETH` via `receive()`. Anyone can call `EthCompoundPCVDeposit.deposit()` to mint `CToken` for the contracts `ETH` balance.

The `CToken` to be used is configured on `EthCompoundPCVDeposit` deployment. It is not checked, whether the provided `CToken` address is actually a valid `CToken`.

If the configured `CToken` ceases to work correctly (e.g. `CToken.mint|redeem*` disabled or the configured `CToken` address is invalid), `ETH` held by the contract may be locked up.

Recommendation

Similar to `EthLidoPCVDeposit` add a method `witdrawETH`, access-restricted to `onlyPCVController`, that allows recovering `ETH` from the `EthCompoundPCVDeposit` contract in case the `CToken` contract throws. (Consider moving this functionality to `PCVDeposit` where `withdrawERC20` is implemented to avoid having to implement this over and over again)

In `CompoundPCVDepositBase` consider verifying, that the `CToken` constructor argument is actually a valid `CToken` by checking `require(ctoken.isCToken(), "not a valid CToken")`.

3.6 TribalChief - Governor decrease of pool's allocation point should unlock depositors' funds Minor

Description

When the `TribalChief` governor decreases the ratio between the allocation point (`PoolInfo.allocPoint`) and the total allocation point (`totalAllocPoint`) for a specific pool (either be directly decreasing `PoolInfo.allocPoint` of a given pool, or by increasing this value for other pools), the total reward for this pool is decreased as well. Depositors should be able to withdraw their funds immediately after this kind of change.

Examples

code/contracts/staking/TribalChief.sol:L252-L261

```
function set(uint256 _pid, uint128 _allocPoint, IRewarder _rewarder, bool overwrite) public onlyGovernor {
    totalAllocPoint = (totalAllocPoint - poolInfo[_pid].allocPoint) + _allocPoint;
    poolInfo[_pid].allocPoint = _allocPoint.toUint64();

    if (overwrite) {
        rewarder[_pid] = _rewarder;
    }

    emit LogSetPool(_pid, _allocPoint, overwrite ? _rewarder : rewarder[_pid], overwrite);
}
```

Recommendation

Make sure that depositors' funds are unlocked for pools that affected negatively by calling `TribalChief.set`.

3.7 TribalChief - new block reward retrospectively takes effect on pools that have not been updated recently Minor

Description

When the governor updates the block reward `tribalChiefTribePerBlock` the new reward is applied for the outstanding duration of blocks in `updatePool`. This means, if a pool hasn't updated in a while (unlikely) the new block reward is retrospectively applied to the pending duration instead of starting from when the block reward changed.

Examples

- rewards calculation

code/contracts/staking/TribalChief.sol:L323-L327

```
if (virtualSupply > 0) {
    uint256 blocks = block.number - pool.lastRewardBlock;
    uint256 tribeReward = (blocks * tribePerBlock() * pool.allocPoint) / totalAllocPoint;
    pool.accTribePerShare = uint128(pool.accTribePerShare + ((tribeReward * ACC_TRIBE_PRECISION) / virtualSupply));
}
```

- updating the block reward

code/contracts/staking/TribalChief.sol:L111-L116

```
/// @notice Allows governor to change the amount of tribe per block
/// @param newBlockReward The new amount of tribe per block to distribute
function updateBlockReward(uint256 newBlockReward) external onlyGovernor {
    tribalChiefTribePerBlock = newBlockReward;
    emit NewTribePerBlock(newBlockReward);
}
```

Recommendation

It is recommended to update pools before changing the block reward. Document and make users aware that the new reward is applied to the outstanding duration when calling `updatePool`.

3.8 TribalChief - duplicate import SafeERC20 Minor

Description

Duplicate import for SafeERC20.

Examples

code/contracts/staking/TribalChief.sol:L7-L8

```
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

Recommendation

Remove duplicate import line.

3.9 TribalChief - resetRewards should emit an event Minor

Description

The method `resetRewards` silently resets a pools tribe allocation.

Examples

code/contracts/staking/TribalChief.sol:L263-L275

```
/// @notice Reset the given pool's TRIBE allocation to 0 and unlock the pool. Can only be called by the governor or guardian.
/// @param _pid The index of the pool. See `poolInfo`.
function resetRewards(uint256 _pid) public onlyGuardianOrGovernor {
    // set the pool's allocation points to zero
    totalAllocPoint = (totalAllocPoint - poolInfo[_pid].allocPoint);
    poolInfo[_pid].allocPoint = 0;

    // unlock all staked tokens in the pool
    poolInfo[_pid].unlocked = true;

    // erase any IRewarder mapping
    rewarder[_pid] = IRewarder(address(0));
}
```

Recommendation

For transparency and to create an easily accessible audit trail of events consider emitting an event when resetting a pools allocation.

4 Recommendations

4.1 EthCompoundPCVDeposit - stick to upstream interface contract names

Recommendation

Stick to the original upstream interface names to make clear with which external system the contract interacts with. Rename `CEth` to `CEther`. See original upstream interface name.

code/contracts/pcv/compound/EthCompoundPCVDeposit.sol:L6-L8

```
interface CEth {
    function mint() external payable;
}
```

4.2 CompoundPCVDepositBase - verify provided CToken address is actually a CToken

Recommendation

The ctoken address provided when deploying a new `*CompoundPCVDeposit` is never validated. Consider adding the following check: `require(_cToken.isCToken, "not a valid CToken")`.

code/contracts/pcv/compound/CompoundPCVDepositBase.sol:L25-L30

```
constructor(
    address _core,
    address _cToken
) CoreRef(_core) {
    cToken = CToken(_cToken);
}
```

4.3 CompoundPCV - documentation & testing

Recommendation

Currently, the PCV flavor is only unit-tested using a mocked `CToken`. Consider providing integration tests that actually integrate and operate it in a compound test environment.

Provide a specification. & documentation describing the roles and functionality of the contract. Who deploys the PCVDeposit contract? Who Deploys the CToken and therefore may be in control of certain adminOnly functions of the CToken? What are the requirements for a CToken to be usable with CompoundPCVDeposit (listed/unlisted, ...)? Who has the potential power to borrow assets on behalf of the collateral provided?

4.4 TribalChief - immutable vs constant

Recommendation

Constant state variables that are not initialized with the constructor can be `constant` instead of `immutable`.

code/contracts/staking/TribalChief.sol:L88-L90

```
uint256 private immutable ACC_TRIBE_PRECISION = 1e12;
/// exponent for rewards multiplier
uint256 public immutable SCALE_FACTOR = 1e18;
```

4.5 TribalChief - governorAddPoolMultiplier should emit a PoolLocked event

Description

Users should be notified if the pool gets unlocked during a call to `governorAddPoolMultiplier`. Consider emitting a `PoolLocked(false)` event.

code/contracts/staking/TribalChief.sol:L143-L158

```
function governorAddPoolMultiplier(
    uint256 _pid,
    uint64 lockLength,
    uint64 newRewardsMultiplier
) external onlyGovernor {
    PoolInfo storage pool = poolInfo[_pid];
    uint256 currentMultiplier = rewardMultipliers[_pid][lockLength];
    // if the new multplier is less than the current multiplier,
    // then, you need to unlock the pool to allow users to withdraw
    if (newRewardsMultiplier < currentMultiplier) {
        pool.unlocked = true;
    }
    rewardMultipliers[_pid][lockLength] = newRewardsMultiplier;

    emit LogPoolMultiplier(_pid, lockLength, newRewardsMultiplier);
}
```

4.6 TribalChief - `updatePool` invocation inside `_harvest` should be moved to `harvest` instead

Description

When `TribalChief.withdrawAllAndHarvest` is executed, there's a redundant invocation of `TribalChief.updatePool` that caused by `TribalChief._harvest`, that can be moved to `TribalChief.harvest` instead.

Examples

code/contracts/staking/TribalChief.sol:L485-L515

```
function _harvest(uint256 pid, address to) private {
    updatePool(pid);
    PoolInfo storage pool = poolInfo[pid];
    UserInfo storage user = userInfo[pid][msg.sender];

    // assumption here is that we will never go over 2^128 -1
    int256 accumulatedTribe = int256( uint256(user.virtualAmount) * uint256(pool.accTribePerShare) ) / int256(ACC_TRIBE_PRECISIO

    // this should never happen
    require(accumulatedTribe >= 0 || (accumulatedTribe - user.rewardDebt) < 0, "negative accumulated tribe");

    uint256 pendingTribe = uint256(accumulatedTribe - user.rewardDebt);

    // if pending tribe is ever negative, revert as this can cause an underflow when we turn this number to a uint
    require(pendingTribe.toInt256() >= 0, "pendingTribe is less than 0");

    // Effects
    user.rewardDebt = int128(accumulatedTribe);

    // Interactions
    if (pendingTribe != 0) {
        TRIBE.safeTransfer(to, pendingTribe);
    }

    IRewarder _rewarder = rewarder[pid];
    if (address(_rewarder) != address(0)) {
        _rewarder.onSushiReward( pid, msg.sender, to, pendingTribe, user.virtualAmount);
    }

    emit Harvest(msg.sender, pid, pendingTribe);
}
```

Appendix 1 - 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.

Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit or a 1-day security review.

CONTACT US

AUDITS

BLOG

TOOLS

RESEARCH

ABOUT

CONTACT

CAREERS

PRIVACY POLICY

Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

e-mail address

POWERED BY CONSENSYS