Fei Protocol v2 Phase 1

Date	September 2021
Auditors	Sergii Kravchenko, Heiko Fisch, Eli Leers, Martin Ortner, Bernhard Gomig

1 Executive Summary

This report presents the results of our engagement with **Fei Protocol** to review **Fei v2 Phase 1**.

1.1 Stage 1

The review was conducted over two weeks, from September 13 – 24, 2021. A total of 30 person-days were spent.

During the first week, the team ramped up on understanding the system and the significant changes that are introduced. These efforts continued into the second week where the team followed up on potential threats to specific components. It should be noted that a two-week engagement is likely not enough for the risk profile and size of the system and that this review is a best effort for the time allotted.

Scope

Our review focused on the commit hash `5e3e2ab889f06831f4fe2e8460066ded40ccf0a8`. The list of files in scope can be found in the Appendix.

1.2 Stage 2

The time spent during the first stage of the review was not enough to sufficiently review the system. Because of that, we dedicated one more week with a limited scope to check some of the most critical properties. The main focus of the review was targeted at the launch of the Tribe buyback pipeline.

Here are some of the critical risks that we are aming to check:

- Minting more FEI than it should be by the `PCVEquityMinter`.
- Converting FEI to Tribe at a wrong price in the Balancer contract.
- Locking up or stealing funds directly from one of the contracts.

Potential risks that were NOT checked:

- Incorrect data from the Collateralization oracle. The `PCVEquityMinter` uses this data to determine the amount of FEI to be minted for the buyback. The main risk is that many FEI tokens will be minted if somebody can attack the system and increase the collateralization rate. The potential attack is mitigated by the fact that there is a cap that limits the maximum amount of minted FEI. We recommend setting this cap to a relatively small amount initially. That will decrease the risk and will help to test the system in practice.
- Balancer contract malfunction. The Balancer contracts were not in the scope due to the time limits. There may be some potential risk related to the Balancer contracts that we did not check in this review.

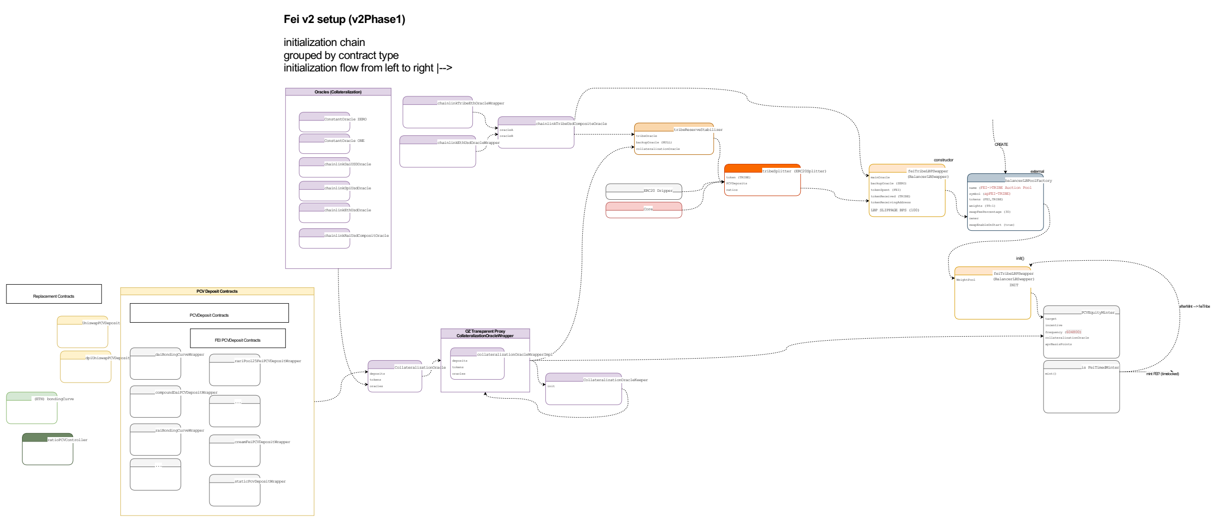
Scope

The second stage of the review was focused on the commit hash `ababe68db266922dda927bf756f44b05dc08f873`. The scope was limited to the following contracts:

- pcv/balancer/*
- token/PCVEquityMinter.sol
- token/FeiTimedMinter.sol
- utils/RateLimitedMinter.sol
- utils/RateLimited.sol
- token/IFeiTimedMinter.sol
- token/IPCVEquityMinter.sol

2 System Overview

The following diagram shows the FEIv2 Phase1 deployment procedure. It outlines from left to right which contracts are instantiated first and whether a contract is initialized with another contracts address. Assuming that if one contract is configured with another contracts address both contracts interact with each other it is possible to derive a high-level interaction diagram that somewhat outlines the flow of data. The purpose of this diagram is to quickly get a high-level understanding of how components may interact with each other. It does not necessarily need to be complete.

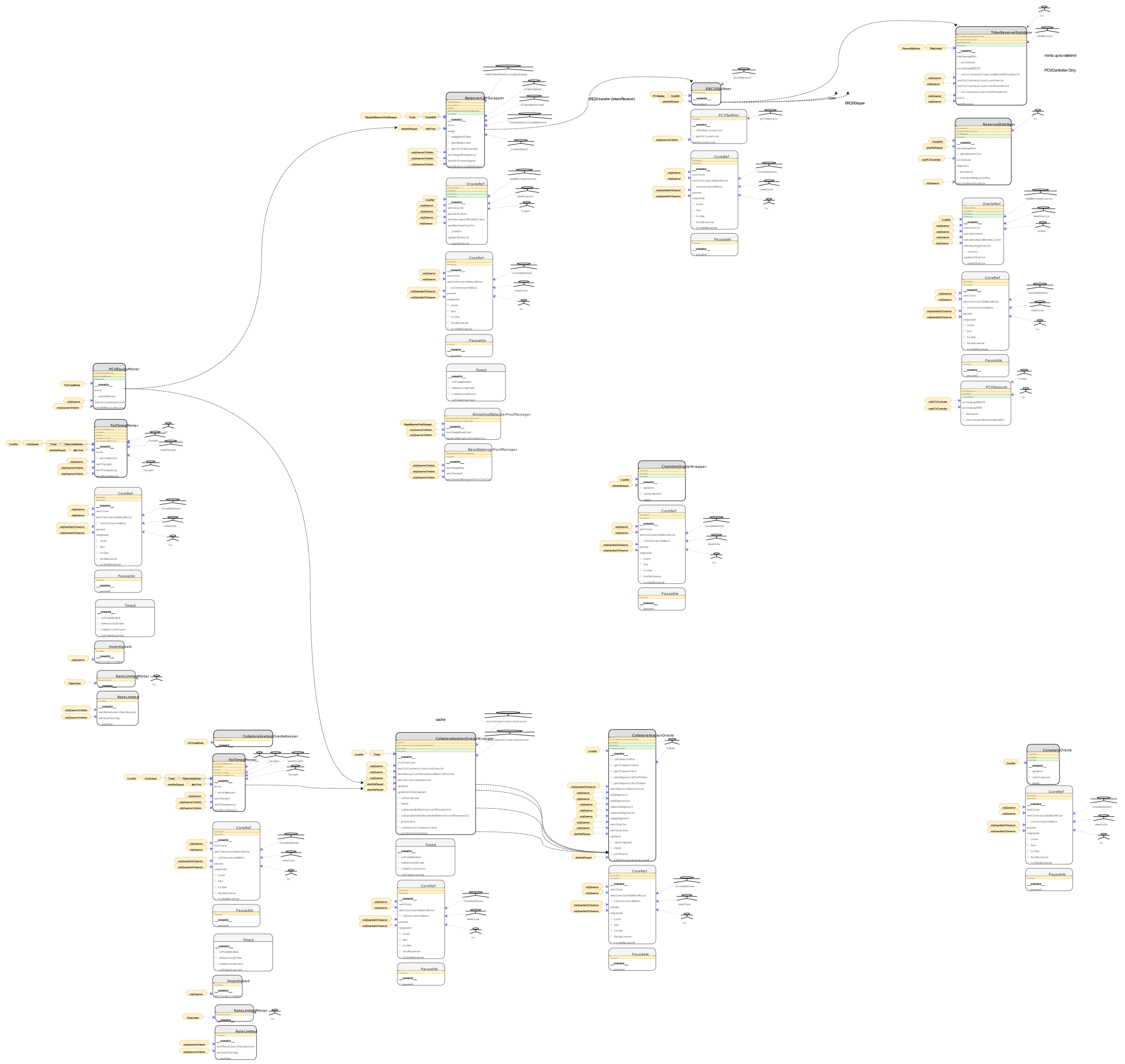


FEI v2 Phase 1 deployment procedure

Main Components:

- PCVDeposits
- OracleWrapper and Composite Oracles
- CollateralizationOracle
- EquityMinter
- FeiTribeLBSwapper
- TribeSplitter
- TribeReserveStabilizer

Another **incomplete** view on the system is provided with the following diagram that depicts high-level contract interaction and reachable contract interfaces. The diagram is not complete due to time constraints, however, we chose to include it as it might help verify the clients model of the system.



FEI v2 Phase 1 high-level contract interaction

Contracts are depicted as boxes. Public reachable interface methods are outlined as rows in the box. The icon indicates that a method is declared as non-state-changing (view/pure) while other methods may change state. A yellow dashed row at the top of the contract shows inherited contracts. A green dashed row at the top of the contract indicates that that contract is used in a usingFor declaration. Modifiers used as ACL are connected as yellow bubbles in front of methods.

3 Stage 2 Findings

This section lists the issues found in the second stage.

3.1 Re-initialization of the Balancer pool is potentially possible `Minor`

Description

Instead of creating a new Balancer pool for an auction every time, the same pool is getting re-used repeatedly. When this happens, the old liquidity is withdrawn, and if there is enough FEI in the contract, the weights are shifted pool is filled with new tokens. If there is not enough FEI, the pool is left empty, and users can still interact with it. When there's enough FEI again, it's re-initialized again, which is not the intention:

code_new/contracts/pcv/balancer/BalancerLBPSwapper.sol:L180-L187

```
uint256 bptTotal = pool.totalSupply();
uint256 bptBalance = pool.balanceOf(address(this));

// Balancer locks a small amount of bptTotal after init, so 0 bpt means pool needs initializing
if (bptTotal == 0) {
    _initializePool();
    return;
}
```

Theoretically, this will never happen because there should be minimal leftover liquidity tokens after the withdrawal. But we couldn't strictly verify that fact because it requires looking into balancer code much deeper.

Recommendation

One of the options would be only to allow re-using the pool in atomic transactions. So if there are not enough FEI tokens for the next auction, the `swap` transaction reverts. That will help with another issue (issue 3.2) too.

3.2 The `BalancerLBPSwapper` may not have enough `Tribe` tokens `Minor`

Description

Whenever the `swap` function is called, it should re-initialize the Balancer pool that requires adding liquidity: 99% Fei and 1% Tribe. So the Tribe should initially be in the contract.

code_new/contracts/pcv/balancer/BalancerLBPSwapper.sol:L313-L325

```
function _getTokensIn(uint256 spentTokenBalance) internal view returns(uint256[] memory amountsIn) {
    amountsIn = new uint256[](2);

    uint256 receivedTokenBalance = readOracle().mul(spentTokenBalance).mul(ONE_PERCENT).div(NINETY_NIN

    if (address(assets[0]) == tokenSpent) {
        amountsIn[0] = spentTokenBalance;
        amountsIn[1] = receivedTokenBalance;
    } else {
        amountsIn[0] = receivedTokenBalance;
        amountsIn[1] = spentTokenBalance;
    }
}
```

Additionally, when the `swap` is called, and there is not enough FEI to re-initiate the Balancer auction, all the Tribe gets withdrawn. So the next time the `swap` is called, there is no Tribe in the contract again.

code_new/contracts/pcv/balancer/BalancerLBPSwapper.sol:L248-L249

```
// 5. Send remaining tokenReceived to target
IERC20(tokenReceived).transfer(tokenReceivingAddress, IERC20(tokenReceived).balanceOf(address(this)));
```

Recommendation

Create an automated mechanism that mints/transfers Tribe when it is needed in the swapper contract.

3.3 No emergency exit strategy for `BalancerLBPSwapper`

Description

If something goes wrong with the balancer contract, there are a lot of functions that prevent people from using the pool or calling the `swap` function of the `BalancerLBPSwapper` contract. But if that happens, no function that withdraws the liquidity from the pool. The liquidity can currently only be withdrawn by the `swap` function, which will probably be paused and has some restrictions.

Recommendation

Add the emergency exit function.

4 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.

4.1 StableSwapOperatorV1 - `resistantFei` value is not correct in the `resistantBalanceAndFei` function `Major`

Description

The `resistantBalanceAndFei` function of a `PCVDeposit` contract is supposed to return the amount of funds that the contract controls; it is then used to evaluate the total value of PCV (collateral in the protocol). Additionally, this function returns the number of FEI tokens that are protocol-controlled. These FEI tokens are “temporarily minted”; they are not backed up by the collateral and shouldn’t be used in calculations that determine the collateralization of the protocol.

Ideally, the amount of these FEI tokens should be the same during the deposit, withdrawal, and the `resistantBalanceAndFei` function call. In the `StableSwapOperatorV1` contract, all these values are totally different:

- during the deposit, the amount of required FEI tokens is calculated. It’s done in a way so the values of FEI and 3pool tokens in the metapool should be equal after the deposit. So if there is the initial imbalance of FEI and 3pool tokens, the deposit value of these tokens will be different:

code/contracts/pcv/curve/StableSwapOperatorV1.sol:L156-L171

```
// get the amount of tokens in the pool
(uint256 _3crvAmount, uint256 _feiAmount) = (
    IStableSwap2(pool).balances(_3crvIndex),
    IStableSwap2(pool).balances(_feiIndex)
);
// ... and the expected amount of 3crv in it after deposit
uint256 _3crvAmountAfter = _3crvAmount + _3crvBalanceAfter;

// get the usd value of 3crv in the pool
uint256 _3crvUsdValue = _3crvAmountAfter * IStableSwap3(_3pool).get_virtual_price() / 1e18;

// compute the number of FEI to deposit
uint256 _feiToDeposit = 0;
if (_3crvUsdValue > _feiAmount) {
    _feiToDeposit = _3crvUsdValue - _feiAmount;
}
```

- during the withdrawal, the FEI and 3pool tokens are withdrawn in the same proportion as they are present in the metapool:

code/contracts/pcv/curve/StableSwapOperatorV1.sol:L255-L258

```
uint256[2] memory _minAmounts; // [0, 0]
IERC20(pool).approve(pool, _lpToWithdraw);
uint256 _3crvBalanceBefore = IERC20(_3crv).balanceOf(address(this));
IStableSwap2(pool).remove_liquidity(_lpToWithdraw, _minAmounts);
```

- in the `resistantBalanceAndFei` function, the value of protocol-controlled FEI tokens and the value of 3pool tokens deposited are considered equal:

code/contracts/pcv/curve/StableSwapOperatorV1.sol:L348-L349

```
resistantBalance = _lpPriceUSD / 2;
resistantFei = resistantBalance;
```

Some of these values may be equal under some circumstances, but that is not enforced. After one of the steps (deposit or withdrawal), the total PCV value and collateralization may be changed significantly.

Recommendation

Make sure that deposit, withdrawal, and the `resistantBalanceAndFei` are consistent and won’t instantly change the PCV value significantly.

4.2 CollateralizationOracle - Fei in excluded deposits contributes to `userCirculatingFei` Major

Description

`CollateralizationOracle.pcvStats` iterates over all deposits, queries the resistant balance and FEI for each deposit, and accumulates the total value of the resistant balances and the total resistant FEI. Any Guardian or Governor can exclude (and re-include) a deposit that has become problematic in some way, for example, because it is reporting wrong numbers. Finally, the `pcvStats` function computes the `userCirculatingFei` as the total FEI supply minus the accumulated resistant FEI balances; the idea here is to determine the amount of “free” FEI, or FEI that is not PCV. However, the FEI balances from excluded deposits contribute to the `userCirculatingFei`, although they are clearly not “free” FEI. That leads to a wrong `protocolEquity` and a skewed collateralization ratio and might therefore have a significant impact on the economics of the system.

It should be noted that even the exclusion from the total PCV leads to a `protocolEquity` and a collateralization ratio that could be considered skewed (again, it might depend on the exact reasons for exclusion), but “adding” the missing FEI to the `userCirculatingFei` distorts these numbers even more.

In the extreme scenario that *all* deposits have been excluded, the entire Fei supply is currently reported as `userCirculatingFei`.

code/contracts/oracle/CollateralizationOracle.sol:L278-L328

```
/// @notice returns the Protocol-Controlled Value, User-circulating FEI, and
///         Protocol Equity.
/// @return protocolControlledValue : the total USD value of all assets held
///         by the protocol.
/// @return userCirculatingFei : the number of FEI not owned by the protocol.
/// @return protocolEquity : the difference between PCV and user circulating FEI.
///         If there are more circulating FEI than $ in the PCV, equity is 0.
/// @return validityStatus : the current oracle validity status (false if any
///         of the oracles for tokens held in the PCV are invalid, or if
///         this contract is paused).
function pcvStats() public override view returns (
  uint256 protocolControlledValue,
  uint256 userCirculatingFei,
  int256 protocolEquity,
  bool validityStatus
) {
    uint256 _protocolControlledFei = 0;
    validityStatus = !paused();

    // For each token...
    for (uint256 i = 0; i < tokensInPcv.length(); i++) {
        address _token = tokensInPcv.at(i);
        uint256 _totalTokenBalance  = 0;

        // For each deposit...
        for (uint256 j = 0; j < tokenToDeposits[_token].length(); j++) {
            address _deposit = tokenToDeposits[_token].at(j);

            // ignore deposits that are excluded by the Guardian
            if (!excludedDeposits[_deposit]) {
                // read the deposit, and increment token balance/protocol fei
                (uint256 _depositBalance, uint256 _depositFei) = IPCVDepositBalances(_deposit).resista
                _totalTokenBalance += _depositBalance;
                _protocolControlledFei += _depositFei;
            }
        }

        // If the protocol holds non-zero balance of tokens, fetch the oracle price to
        // increment PCV by _totalTokenBalance * oracle price USD.
        if (_totalTokenBalance != 0) {
            (Decimal.D256 memory _oraclePrice, bool _oracleValid) = IOracle(tokenToOracle[_token]).rea
            if (!_oracleValid) {
                validityStatus = false;
            }
            protocolControlledValue += _oraclePrice.mul(_totalTokenBalance).asUint256();
        }
    }

    userCirculatingFei = fei().totalSupply() - _protocolControlledFei;
    protocolEquity = int256(protocolControlledValue) - int256(userCirculatingFei);
}
```

Recommendation

It is unclear how to fix this. One might want to exclude the FEI in excluded deposits *entirely* from the calculation, but not knowing the amount was the reason to exclude the deposit in the first place.

One option could be to let the entity that excludes a deposit specify substitute values that should be used instead of querying the numbers from the deposit. However, it is questionable whether this approach is practical if the numbers we'd like to see as substitute values change quickly or repeatedly over time. Ultimately, the querying function itself should be fixed. Moreover, as the substitute values can dramatically impact the system economics, we'd only like to trust the Governor with this and not give this permission to a Guardian. However, the original intention was to give a role with less trust than the Governor the possibility to react quickly to a deposit that reports wrong numbers; if the exclusion of deposits becomes the Governor's privilege, such a quick and lightweight intervention isn't possible anymore.

Independently, we recommend taking proper care of the situation that *all* deposits - or just too many - have been excluded, for example, by setting the returned `validityStatus` to `false`, as in this case, there is not enough information to compute the collateralization ratio even as a crude approximation.

4.3 StableSwapOperatorV1 - the `_minLpOut` value is not accurate Medium

Description

When depositing, the expected minimum amount of the output LP tokens is calculated:

code/contracts/pcv/curve/StableSwapOperatorV1.sol:L194-L200

```
// slippage check on metapool deposit
uint256 _balanceDeposited = IERC20(pool).balanceOf(address(this)) - _balanceBefore;
{
    uint256 _metapoolVirtualPrice = IStableSwap2(pool).get_virtual_price();
    uint256 _minLpOut = (_feiToDeposit + _3crvBalanceAfter) * 1e18 / _metapoolVirtualPrice * (Constant
    require(_balanceDeposited >= _minLpOut, "StableSwapOperatorV1: metapool deposit slippage too high"
}
```

The problem is that the `get_virtual_price` function returns a valid price only if the tokens in the pool are expected to have a price equal to $1 which is not the case. Also, the balances of deposited FEI and 3pool lp tokens are just added to each other while they have a different price: `_feiToDeposit + _3crvBalanceAfter`.

The price of the 3pool lp tokens is currently very close to 1$ so this difference is not that visible at the moment, but this can slowly change over time.

4.4 StableSwapOperatorV1 - FEI tokens in the contract are not considerred as protocol-owned Medium

Description

Every `PCVDeposit` contract should return the amount of PCV controlled by this contract in the `resistantBalanceAndFei`. In addition to that, this function returns the amount of protocol-controlled FEI, which is not supposed to be collateralized. These values are crucial for evaluating the collateralization of the protocol.

Unlike some other `PCVDeposit` contracts, protocol-controlled FEI is not minted during the deposit and not burnt during the withdrawal. These FEI tokens are transferred beforehand, so when depositing, all the FEI that are instantly becoming protocol-controlled and heavily impact the collateralization rate. The opposite impact, but as much significant, happens during the withdrawal.

The amount of FEI needed for the deposited is calculated dynamically, it is hard to predict the exact amount beforehand. There may be too many FEI tokens in the contract and the leftovers will be considered as the user-controlled FEI.

Recommendation

There may be different approaches to solve this issue. One of them would be to make sure that the Fei transfers to/from the contract and the deposit/withdraw calls are happening in a single transaction. These FEI should be minted, burnt, or re-used as the protocol-controlled FEI in the same transaction. Another option would be to consider all the FEI balance in the contract as the protocol-controlled FEI.

If the intention is to have all these FEI collateralized, the other solution is needed: make sure that `resistantBalanceAndFei` always returns `resistantFei` equals zero.

4.5 BalancerLBPSwapper - init() can be front-run to potentially steal tokens `Medium`

Description

The deployment process for `BalancerLBPSwapper` appears to be the following:

1. deploy `BalancerLBPSwapper`.
2. run `ILiquidityBootstrappingPoolFactory.create()` proving the newly deployed swapper address as the owner of the pool.
3. initialize `BalancerLBPSwapper.init()` with the address of the newly created pool.

This process may be split across multiple transactions as in the `v2Phase1.js` deployment scenario.

Between step (1) and (3) there is a window of opportunity for someone to maliciously initialize contract. This should be easily detectable because calling `init()` twice should revert the second transaction. If this is not caught in the deployment script this may have more severe security implications. Otherwise, this window can be used to grief the deployment initializing it before the original initializer does forcing them to redeploy the contract or to steal any `tokenSpent`/`tokenReceived` that are owned by the contract at this time.

Note: It is assumed that the contract will not own a lot of tokens right after deployment rendering the scenario of stealing tokens more unlikely. However, that highly depends on the deployment script for the contract system.

Examples

code/contracts/pcv/balancer/BalancerLBPSwapper.sol:L107-L117

```
function init(IWeightedPool _pool) external {
    require(address(pool) == address(0), "BalancerLBPSwapper: initialized");

    pool = _pool;
    IVault _vault = _pool.getVault();

    vault = _vault;

    // Check ownership
    require(_pool.getOwner() == address(this), "BalancerLBPSwapper: contract not pool owner");
```

code/contracts/pcv/balancer/BalancerLBPSwapper.sol:L159-L160

```
IERC20(tokenSpent).approve(address(_vault), type(uint256).max);
IERC20(tokenReceived).approve(address(_vault), type(uint256).max);
```

Recommendation

protect `BalancerLBPSwapper.init()` and only allow a trusted entity (e.g. the initial deployer) to call this method.

4.6 PCVEquityMinter and BalancerLBPSwapper - desynchronisation race Medium

Description

There is nothing that prevents other actors from calling `BalancerLBPSwapper.swap()` `afterTime` but right before `PCVEquityMinter.mint()` would as long as the `minAmount` required for the call to pass is deposited to `BalancerLBPSwapper`.

Both the `PCVEquityMinter.mint()` and `BalancerLBPSwapper.swap()` are timed (via the `afterTime` modifier) and are ideally in sync. In an ideal world the incentive to call `mint()` would be enough to ensure that both contracts are always in sync, however, a malicious actor might interfere by calling `.swap()` directly, providing the `minAmount` required for the call to pass. This will have two effects:

- instead of taking the newly minted FEI from `PCVEquityMinter`, existing FEI from the malicious user will be used with the pool. (instead of inflating the token the malicious actor basically pays for it)
- the `Timed` modifiers of both contracts will be out of sync with `BalancerLBPSwapper.swap()` being reset (and failing until it becomes available again) and `PCVEquityMinter.mint()` still being available. Furthermore, keeper-scripts (or actors that want to get the incentive) might continue to attempt to `mint()` while the call will ultimately fail in `.swap()` due to the resynchronization of `timed` (unless they simulate the calls first).

Note: There are not a lot of incentives to actually exploit this other than preventing protocol inflation (mint) and potentially griefing users. A malicious user will lose out on the incentivized call and has to ensure that the `minAmount` required for `.swap()` to work is available. It is, however, in the best interest of security to defuse the unpredictable racy character of the contract interaction.

Examples

code/contracts/token/PCVEquityMinter.sol:L91-L93

```
function _afterMint() internal override {
    IPCVSwapper(target).swap();
}
```

code/contracts/pcv/balancer/BalancerLBPSwapper.sol:L172-L181

```
function swap() external override afterTime whenNotPaused {
    (
        uint256 spentReserves,
        uint256 receivedReserves,
        uint256 lastChangeBlock
    ) = getReserves();

    // Ensures no actor can change the pool contents earlier in the block
    require(lastChangeBlock < block.number, "BalancerLBPSwapper: pool changed this block");
```

Recommendation

If `BalancerLBPSwapper.swap()` is only to be called within the flows of action from a `PCVEquityMinter.mint()` it is suggested to authenticate the call and only let `PCVEquityMinter` call `.swap()`

4.7 CollateralizationOracleWrapper - the deviation threshold check in `update()` always returns false Medium

Description

A call to `update()` returns a boolean flag indicating whether the update was performed on outdated data. This flag is being checked in `updateIfOutdated()` which is typically called by an incentivized keeper function.

The `_isExceededDeviationThreshold` calls at the end of the `_update()` function always return `false` as they are comparing the same values (`cachedProtocolControlledValue` to the `_protocolControlledValue` value and `cachedProtocolControlledValue` has just been set to `_protocolControlledValue` a couple of lines before). `_isExceededDeviationThreshold` will, therefore, never detect a deviation and return `false'.

There may currently be no incentive (e.g. from the keeper side) to call `update()` if the values are not outdated but they deviated too much from the target. However, anyone can force an update by calling the non-incentivized public `update()` method instead.

Examples

code/contracts/oracle/CollateralizationOracleWrapper.sol:L156-L177

```
    require(_validityStatus, "CollateralizationOracleWrapper: CollateralizationOracle is invalid");

    // set cache variables
    cachedProtocolControlledValue = _protocolControlledValue;
    cachedUserCirculatingFei = _userCirculatingFei;
    cachedProtocolEquity = _protocolEquity;

    // reset time
    _initTimed();

    // emit event
    emit CachedValueUpdate(
        msg.sender,
        cachedProtocolControlledValue,
        cachedUserCirculatingFei,
        cachedProtocolEquity
    );

    return outdated
        || _isExceededDeviationThreshold(cachedProtocolControlledValue, _protocolControlledValue)
        || _isExceededDeviationThreshold(cachedUserCirculatingFei, _userCirculatingFei);
}
```

Recommendation

- Add unit tests to check for all three return conditions (timed, deviationA, deviationB)
- Make sure to compare the current to the stored value before updating the cached values when calling `_isExceededDeviationThreshold`.

4.8 ChainlinkOracleWrapper - latestRoundData might return stale results Medium

Description

The oracle wrapper calls out to a chainlink oracle receiving the `latestRoundData()`. It then checks freshness by verifying that the answer is indeed for the last known round. The returned `updatedAt` timestamp is not checked.

If there is a problem with chainlink starting a new round and finding consensus on the new value for the oracle (e.g. chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the chainlink system) consumers of this contract may continue using outdated stale data (if oracles are unable to submit no new round is started)

Examples

code/contracts/oracle/ChainlinkOracleWrapper.sol:L49-L58

```
/// @notice read the oracle price
/// @return oracle price
/// @return true if price is valid
function read() external view override returns (Decimal.D256 memory, bool) {
    (uint80 roundId, int256 price,,, uint80 answeredInRound) = chainlinkOracle.latestRoundData();
    bool valid = !paused() && price > 0 && answeredInRound == roundId;

    Decimal.D256 memory value = Decimal.from(uint256(price)).div(oracleDecimalsNormalizer);
    return (value, valid);
}
```

code/contracts/oracle/ChainlinkOracleWrapper.sol:L42-L47

```
/// @notice determine if read value is stale
/// @return true if read value is stale
function isOutdated() external view override returns (bool) {
    (uint80 roundId,,,, uint80 answeredInRound) = chainlinkOracle.latestRoundData();
    return answeredInRound != roundId;
}
```

Recommendation

Consider checking the oracle responses `updatedAt` value after calling out to `chainlinkOracle.latestRoundData()` verifying that the result is within an allowed margin of freshness.

4.9 CollateralizationOracle - missing events and incomplete event information Minor

Description

The `CollateralizationOracle.setDepositExclusion` function is used to exclude and re-include deposits from collateralization calculations. Unlike the other state-changing functions in this contract, it doesn't emit an event to inform about the exclusion or re-inclusion.

code/contracts/oracle/CollateralizationOracle.sol:L111-L113

```
function setDepositExclusion(address _deposit, bool _excluded) external onlyGuardianOrGovernor {
    excludedDeposits[_deposit] = _excluded;
}
```

The `DepositAdd` event emits not only the deposit address but also the deposit's token. Despite the symmetry, the `DepositRemove` event does not emit the token.

code/contracts/oracle/CollateralizationOracle.sol:L25-L26

```
event DepositAdd(address from, address indexed deposit, address indexed token);
event DepositRemove(address from, address indexed deposit);
```

Recommendation

1. `setDepositInclusion` should emit an event that informs about the deposit and whether it was

included or excluded.

2. For symmetry reasons and because it is indeed useful information, the `DepositRemove` event could include the deposit's token.

4.10 RateLimited - Contract starts with a full buffer at deployment Minor

Description

A contract that inherits from `RateLimited` starts out with a full buffer when it is deployed.

code/contracts/utils/RateLimited.sol:L35

```
_bufferStored = _bufferCap;
```

That means the full `bufferCap` is immediately available after deployment; it doesn't have to be built up over time. This behavior might be unexpected.

Recommendation

We recommend starting with an empty buffer, or - if there are valid reasons for the current implementation - at least document it clearly.

4.11 StableSwapOperatorV1 - the contract relies on the 1$ price of every token in 3pool Minor

Description

To evaluate the price of the 3pool lp token, the built-in `get_virtual_price` function is used. This function is supposed to be a manipulation-resistant pricing function that works under the assumption that all the tokens in the pool are worth 1$. If one of the tokens is broken and is priced less, the price is harder to calculate. For example, Chainlink uses the following function to calculate at least the lower boundary of the lp price: https://blog.chain.link/using-chainlink-oracles-to-securely-utilize-curve-lp-pools/

The withdrawal and the controlled value calculation are always made in DAI instead of other stablecoins of the 3pool. So if DAI gets compromised but other tokens aren't, there is no way to switch to them.

4.12 BalancerLBPSwapper - tokenSpent and tokenReceived should be immutable Minor

Description

Acc. to the inline comment both `tokenSpent` and `tokenReceived` should be immutable but they are not declared as such.

Examples

code/contracts/pcv/balancer/BalancerLBPSwapper.sol:L92-L94

```
// tokenSpent and tokenReceived are immutable
tokenSpent = _tokenSpent;
tokenReceived = _tokenReceived;
```

code/contracts/pcv/balancer/BalancerLBPSwapper.sol:L40-L44

```
/// @notice the token to be auctioned
address public override tokenSpent;

/// @notice the token to buy
address public override tokenReceived;
```

Recommendation

Declare both variable `immutable`.

4.13 CollateralizationOracle - potentially unsafe casts Minor

Description

`protocolControlledValue` is the cumulative USD token value of all tokens in the PCV. The USD value is determined using external chainlink oracles. To mitigate some effects of attacks on chainlink to propagate to this protocol it is recommended to implement a defensive approach to handling values derived from the external source. Arithm. overflows are checked by the compiler (`0.8.4`), however, it does not guarantee safe casting from unsigned to signed integer. The scenario of this happening might be rather unlikely, however, there is no guarantee that the external price-feed is not taken over by malicious actors and this is when every line of defense counts.

```
//solidity 0.8.7
 » int(uint(2**255))
-57896044618658097711785492504343953926634992332820282019728792003956564819968
 » int(uint(2**255-2))
57896044618658097711785492504343953926634992332820282019728792003956564819966
```

Examples

code/contracts/oracle/CollateralizationOracle.sol:L327-L327

```
protocolEquity = int256(protocolControlledValue) - int256(userCirculatingFei);
```

code/contracts/oracle/CollateralizationOracle.sol:L322-L322

```
protocolControlledValue += _oraclePrice.mul(_totalTokenBalance).asUint256();
```

Recommendation

Perform overflow checked SafeCast as another line of defense against oracle manipulation.

4.14 FeiTimedMinter - constructor does not enforce the same

boundaries as setter for frequency Minor

Description

The setter method for `frequency` enforced upper and lower bounds while the constructor does not. Users cannot trust that the `frequency` is actually set to be within bounds on deployment.

Examples

code/contracts/token/FeiTimedMinter.sol:L32-L48

```
constructor(
    address _core,
    address _target,
    uint256 _incentive,
    uint256 _frequency,
    uint256 _initialMintAmount
)
    CoreRef(_core)
    Timed(_frequency)
    Incentivized(_incentive)
    RateLimitedMinter((_initialMintAmount + _incentive) / _frequency, (_initialMintAmount + _incentive
{
    _initTimed();

    _setTarget(_target);
    _setMintAmount(_initialMintAmount);
}
```

code/contracts/token/FeiTimedMinter.sol:L82-L87

```
function setFrequency(uint256 newFrequency) external override onlyGovernorOrAdmin {
    require(newFrequency >= MIN_MINT_FREQUENCY, "FeiTimedMinter: frequency low");
    require(newFrequency <= MAX_MINT_FREQUENCY, "FeiTimedMinter: frequency high");

    _setDuration(newFrequency);
}
```

Recommendation

Perform the same checks on `frequency` in the constructor as in the `setFrequency` method.

This contract is also inherited by a range of contracts that might specify different boundaries to what is hardcoded in the `FeiTimedMinter`. A way to enforce bounds-checks could be to allow overriding the setter method and using the setter in the constructor as well ensuring that bounds are also checked on deployment.

4.15 CollateralizationOracle - swapDeposit should call internal functions to remove/add deposits Minor

Description

Instead of calling `removeDeposit` and `addDeposit`, `swapDeposit` should call its internal sister functions `_removeDeposit` and `_addDeposit` to avoid running the `onlyGovernor` checks multiple times.

Examples

code/contracts/oracle/CollateralizationOracle.sol:L191-L198

```
/// @notice Swap a PCVDeposit with a new one, for instance when a new version
///         of a deposit (holding the same token) is deployed.
/// @param _oldDeposit : the PCVDeposit to remove from the list.
/// @param _newDeposit : the PCVDeposit to add to the list.
function swapDeposit(address _oldDeposit, address _newDeposit) external onlyGovernor {
    removeDeposit(_oldDeposit);
    addDeposit(_newDeposit);
}
```

Recommendation

Call the internal functions instead. `addDeposit`’s and `removeDeposit`’s visibility can then be changed from `public` to `external`.

4.16 CollateralizationOracle - misleading comments Minor

Description

According to an inline comment in `isOvercollateralized`, the validity status of `pcvStats` is ignored, while it is actually being checked.

Similarly, a comment in `pcvStats` mentions that the returned `protocolEquity` is 0 if there is less PCV than circulating FEI, while in reality, `pcvStats` always returns the difference between the former and the latter, even if it is negative.

Examples

code/contracts/oracle/CollateralizationOracle.sol:L332-L339

```
///         Controlled Value) than the circulating (user-owned) FEI, i.e.
///         a positive Protocol Equity.
///         Note: the validity status is ignored in this function.
function isOvercollateralized() external override view whenNotPaused returns (bool) {
    (,, int256 _protocolEquity, bool _valid) = pcvStats();
    require(_valid, "CollateralizationOracle: reading is invalid");
    return _protocolEquity > 0;
}
```

code/contracts/oracle/CollateralizationOracle.sol:L283-L284

```
/// @return protocolEquity : the difference between PCV and user circulating FEI.
///         If there are more circulating FEI than $ in the PCV, equity is 0.
```

code/contracts/oracle/CollateralizationOracle.sol:L327

```
protocolEquity = int256(protocolControlledValue) - int256(userCirculatingFei);
```

Recommendation

Revise the comments.

5 Recommendations

5.1 Update Natspec

Examples

- `token` is not in natspec

code/contracts/pcv/utils/ERC20Splitter.sol:L6-L28

```
/// @notice a contract to split token held to multiple locations
contract ERC20Splitter is PCVSplitter {

    /// @notice token to split
    IERC20 public token;

    /**
        @notice constructor for ERC20Splitter
        @param _core the Core address to reference
        @param _pcvDeposits the locations to send tokens
        @param _ratios the relative ratios of how much tokens to send each location, in basis points
    */
    constructor(
        address _core,
        IERC20 _token,
        address[] memory _pcvDeposits,
        uint256[] memory _ratios
    )
        CoreRef(_core)
        PCVSplitter(_pcvDeposits, _ratios)
    {
        token = _token;
    }
```

5.2 TribeReserveStabilizer - different minting procedures

Description

The TRIBE token doesn't have a burn functionality. TRIBE that is supposed to be taken out of circulation is sent to the `TribeReserveStabilizer` contract, and when that contract has to mint new TRIBE in exchange for FEI, it will first use up the currently held TRIBE balance before actually minting new tokens.

code/contracts/stabilizer/TribeReserveStabilizer.sol:L117-L133

```
// Transfer held TRIBE first, then mint to cover remainder
function _transfer(address to, uint256 amount) internal override {
    _depleteBuffer(amount);
    uint256 _tribeBalance = balance();
    uint256 mintAmount = amount;
    if(_tribeBalance != 0) {
        uint256 transferAmount = Math.min(_tribeBalance, amount);

        _withdrawERC20(address(token), to, transferAmount);

        mintAmount = mintAmount - transferAmount;
        assert(mintAmount + transferAmount == amount);
    }
    if (mintAmount != 0) {
        _mint(to, mintAmount);
    }
}
```

The contract also has a `mint` function that allows the Governor to mint new TRIBE. Unlike the `exchangeFei` function described above, this function does not first utilize TRIBE held in the contract but directly instructs the token contract to mint the entire amount.

code/contracts/stabilizer/TribeReserveStabilizer.sol:L102-L107

```
/// @notice mints TRIBE to the target address
/// @param to the address to send TRIBE to
/// @param amount the amount of TRIBE to send
function mint(address to, uint256 amount) external override onlyGovernor {
    _mint(to, amount);
}
```

code/contracts/stabilizer/TribeReserveStabilizer.sol:L135-L138

```
function _mint(address to, uint256 amount) internal {
    ITribe _tribe = ITribe(address(token));
    _tribe.mint(to, amount);
}
```

Recommendation

It would make sense and be more consistent with `exchangeFei` if the `mint` function first used TRIBE held in the contract before actually minting new tokens.

Appendix 1 - Files in Scope

Source repository: fei-protocol-core@5e3e2ab889f06831f4fe2e8460066ded40ccf0a8

List of files in scope provided by the client:

File Name	SHA-1 Hash
contracts/Constants.sol	c902dd6cc8084b154f53cc42bca7d8f5c4bb9c51

File Name	SHA-1 Hash
contracts/keeper/CollateralizationOracleKeeper.sol	9f7c84b45a38cbd7383745596a31a5f348e13ccd
contracts/oracle/CollateralizationOracle.sol	97e0248f1ad4114f33f276bedea10f43fd627756
contracts/oracle/CollateralizationOracleWrapper.sol	e5ff4a641db4e5c15baebac03a20848ead0475cd
contracts/oracle/ConstantOracle.sol	8fb90558b299e90a404eaeb405b6e9e7f1940fb9
contracts/oracle/ICollateralizationOracle.sol	c8dd440c92d985801e863b15e054abe5b248480b
contracts/oracle/ICollateralizationOracleWrapper.sol	fa92d5ae07c159b51742e015a9094066e6320d39
contracts/pcv/balancer/manager/WeightedBalancerPoolManager.sol	0a5cec28c830d26b2a0c2bbc52615813179b9bf0
contracts/pcv/balancer/manager/IWeightedBalancerPoolManager.sol	203603756e86f5584a72b75ac5cefeb1207139d7
contracts/pcv/balancer/manager/IBaseBalancerPoolManager.sol	6b51337b07b9769976576a9e051b18ebab5772cd
contracts/pcv/balancer/manager/BaseBalancerPoolManager.sol	c6cd3164a8453cbc83676761297abe90c3fda6bb
contracts/pcv/balancer/IWeightedPool.sol	44b5c11fa53c9f6b9a2980b1f72c0cd80e93a6a0
contracts/pcv/balancer/IVault.sol	830b94d6920c0269ac9db1f4bcf440db34465a83
contracts/pcv/balancer/IBasePool.sol	c46e23c1922bbd66bea59b50f635163d324bb2b5
contracts/pcv/balancer/IAssetManager.sol	9638de75339b3303a055bf54ba951b6a8bc66e1f
contracts/pcv/balancer/BalancerLBPSwapper.sol	aec669380dcb288a127aa5ba8dd9c5981afd4018
contracts/pcv/curve/StableSwapOperatorV1.sol	ca4f876057f79bd84f44340df2d1122f69c00523
contracts/pcv/uniswap/UniswapPCVDeposit.sol#resistantBalanceAndFei()	a4e3117466b65275ee3a25e471f4b256d3b52568
contracts/pcv/utils/ERC20Splitter.sol	788c8c99f73d18bdce2b49bd088c718e1e22d193
contracts/pcv/utils/PCVDepositWrapper.sol	99405a328078280c920d89251ea607489554a715
contracts/pcv/utils/StaticPCVDepositWrapper.sol	2cdedfbf23c4fbc13bb869f6995c23f

9f0097412

File Name	SHA-1 Hash
contracts/stabilizer/TribeReserveStabilizer.sol	cc9fd79907a3ff402f038a6fa05820b82f54bb08
contracts/token/FeiTimedMinter.sol	42f6430cc498a780723cd390ff85379f3ba6cf1d
contracts/token/PCVEquityMinter.sol	d3449e6b6237c3df83872ff0135b4042a6c0a9da
contracts/token/IFeiTimedMinter.sol	dac566c40b65e8f865917a31bc3bc008a9fbf35a
contracts/token/IPCVEquityMinter.sol	ae2af541d55fe164e7fdd7b8d327927dd5f0eeab
contracts/utils/RateLimited.sol	679f4f7303dc8d632f67f57e876f95abd944722f
contracts/utils/RateLimitedMinter.sol	8b73ec82f5a5cef8cb95eb6f0dd06fd6eb2b080d

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.