Fei Protocol Audit – Phase 2

OPENZEPPELIN SECURITY | JULY 2, 2021

Security Audits

The Fei Protocol team asked us to review and audit updates to their protocol smart contracts. This engagement builds upon a previous audit as well as a post-mortem investigation we performed for their protocol. We looked at this new code and now publish our results.

Scope

We audited commit `f54d7bb07c55adb78e2e142e7044f60090bb7602` of the `fei-protocol/fei-protocol-core-internal` repository. It is the intent that these updated contracts will be integrated into the Fei Protocol by their governance process.

In scope were the following contracts:

```
bondingcurve/BondingCurve.sol
bondingcurve/EthBondingCurve.sol
bondingcurve/IBondingCurve.sol
oracle/CompositeOracle.sol
pcv/IPCVDeposit.sol
pcv/IPCVDripController.sol
pcv/IPCVSwapper.sol
pcv/IUniswapPCVController.sol
pcv/IUniswapPCVDeposit.sol
pcv/PCVDripController.sol
pcv/PCVSplitter.sol
pcv/PCVSwapperUniswap.sol
```

```
refs/CoreRef.sol
refs/ICoreRef.sol
refs/IOracleRef.sol
refs/IUniRef.sol
refs/OracleRef.sol
refs/UniRef.sol
stabilizer/EthReserveStabilizer.sol
stabilizer/IReserveStabilizer.sol
stabilizer/ITribeReserveStabilizer.sol
stabilizer/ReserveStabilizer.sol
stabilizer/TribeReserveStabilizer.sol
utils/Incentivized.sol
utils/Timed.sol
```

Note: the repository in question is private at the time of writing, so many hyperlinks will only work for the Fei team.

High level delta of changes

- Removal of genesis, direct incentives, and orchestrator code
- Updated contracts for ERC20 support
- Added a chainlink oracle “oracle/ChainlinkOracleWrapper.sol” (out of scope)
- Simplified Bonding Curve, fixed discount before scale and fixed premium after
- Updated reweight algorithm to be two-sided, and use rebasing below the peg
- Added ReserveStabilizers which sell PCV for FEI at a fixed price <= $1 – FEI, TRIBE, Core, and the DAO remain unchanged, although there are no FEI incentive contracts planned for use We consider the major updates to be the addition of ERC20 support, the simplified Bonding Curve, and the ReserveStabilizer.

ERC20 Support

Previously Fei Protocol only supported the use of ETH as Protocol Controlled Value (PCV). The BondingCurve allowed FEI to be purchased for ETH, and a Uniswap ETH/FEI pool was maintain by the protocol to help keep the FEI price stable. In this updated version of the protocol, the contracts have been rewritten to allow for generic ERC20 PCV throughout the system. A new

Simplified Bonding Curve

In the previous version of the Fei Protocol, the amount of FEI received for a purchase on the BondingCurve contract was calculated using a formula that included (1) how much FEI had been purchased already, and (2) how large the `scale` cap on the curve was. This calculation has now been simplified such that all purchases before `scale` get the same fixed price of `peg-discount`, and once `scale` has been hit all purchases are at a price of `peg+buffer`.

ReserveStabilizers

The ReserveStabilizers provide price support for FEI below the peg by exposing an `exchangeFei` function exchanging discounted FEI for PCV from the held reserves. This way, arbitrageurs can absorb FEI trading below peg on external markets and exchange it for a profit with the ReserveStabilizers. The ReserveStabilizers will be deployed by governance to hold reserves in various ERC20 tokens. There are two main mechanisms in place to automate the flow of PCV into the ReserveStabilizers: – The incentivized `external` `allocate` function of the BondingCurves will distribute their respective PCV amongst set `PCVDeposits`, some of which will be ReserveStabilizers. – The incentivized `external` `drip` function of the `PCVDripController` transfers PCV between `PCVDeposits`. `PCVDripController`s can be deployed and set by governance, where the target of the drip can be a ReserveStabilizer.

Privileged roles

Since the initial audit, there has been a reassignment of roles to the `Timelock` contract: – The `Timelock` contract is a now a Governor, PCVController, and a Minter. Upon deployment of this upgrade, the following roles are planned to be set:

- `EthBondingCurve` and `BondingCurve`'s will have Minter role.
- `PCVDripController` will have PCVController role.
- `PCVSwapperUniswap` will have Minter role.
- `RatioPCVController` will have PCVController role.
- `UniswapPCVController` will have Minter, and Burner roles.
- `UniswapPCVDeposit` will have Minter, and Burner roles.
- `EthReserveStabilizer`, and `ReserveStabilizer` will have Burner role.

The Fei protocol access control system is limited to 5 roles: Minter, Burner, PCVController, Governor, and Guardian. In the words of the Fei Protocol team:

> We assume none of these roles are malicious, and only the Guardian will ever be held by a multisig, the others by contracts. The roles are maintained by Core which is out of scope. There is also a special role “Tribe Minter” which can only be held by a single contract and has the ability to mint TRIBE. It is currently held by the Timelock (managed by Fei DAO/GovernorAlpha) and will be transitioned to the TribeReserveStabilizer. The primary oracle will be either the ChainlinkOracleWrapper or UniswapOracle depending on community preference. Neither is in scope for the audit, both are assumed to be accurate and not malicious or manipulable.

It must be noted that given the Fei documentation stating that the “Tribe Minter” role can only be held by a single contract, when this role is transitioned to the `TribeReserveStabilizer` this “Tribe Minter” role will effectively be shared between the `TribeReserveStabilizer` and governance. We consider this discrepancy between the documentation and implementation an issue and it is covered in our report.

In contrast to early drafts of their whitepaper, where TRIBE had a hard cap, the TRIBE token is now designed to be inflationary without a cap.

Most contracts relying on an oracle, do so through `OracleRef` which has the added benefit of a backup oracle. The lone contract that doesn’t use a backup oracle is the `TribeReserveStabilizer`.

Also, it is worth noting that the rebasing mechanism of the `UniswapPCVController` triggered by the `reweight` function actually disincentivizes liquidity providers, whom are not FEI protocol contracts, from entering the Uniswap FEI liquidity pools. This disincentive comes from the fact that this rebasing mechanism burns FEI from the corresponding Uniswap FEI liquidity pool so that liquidity providers will always suffer a loss in combined liquidity when they exit the pool. Since this is a feature and not a bug, we are sharing this fact here so that the community is aware but we are not otherwise considering it an issue.

Findings

Critical Severity

[C01] Protocol does not support token decimals

This iteration of the Fei Protocol is designed to be a generic version of the original protocol – now instead of supporting just ETH as the PCV of the system, the contracts are designed to be able to use any ERC20 token as PCV. However, the code is not designed to handle tokens that have different numbers of `decimals`, and many of the contracts break in their entirety when a token is used with a different number of tokens than 18. Examples of widely used tokens with a different number of tokens include WBTC (8), USDC (6), and USDT (6).

The root of the issue stems from the fact that the `readOracle` function returns the FEI price as *FEI per X* where FEI and X are both *whole tokens* not token units. Due to the fact that Solidity actually handles tokens in their smallest units, and not as whole tokens, when this price is used for a token with a differing number of decimals to FEI, the calculation result is out by `10 ** |feiDecimals - tokenDecimals|`.

Below we explore some ways in which this affects the protocol. Throughout we use an example where the `token` being used as PCV is WBTC with 8 decimals. A price of 30,000 WBTC/FEI is used.

Exchanging FEI on the `ReserveStabilizer`

The `ReserveStabilizer` contract allows users to burn FEI in exchange for tokens. The `exchangeFei` function calls the `getAmountOut` function to calculate how many tokens the user should receive for their FEI. However `getAmountOut` uses the `readOracle` function without adjusting for the number of decimals that `token` has.

In our example, the WBTC `ReserveStabilizer` would allow a user to deposit 1 FEI (worth ~1 USD), and receive over 300,000 WBTC, worth over $10 billion. This would allow a user to drain all `token` PCV from the contract for a fraction of a dollar.

Purchasing on the `BondingCurve`

Each instantiation of the `BondingCurve` contract allows users to purchase FEI in exchange for a specified ERC20 `token`. The `purchase` function allows the user to specify how many

In our example WBTC `BondingCurve`, a user could provide 1 WBTC to the `BondingCurve.purchase` function before scale, which is specified as `1 * 10**8`. The contract then calculates the amount of FEI to return as `31,579 * 10**8`, which given that FEI has 18 decimals is significantly less than 1 FEI. In this case the user received less than $1 of FEI in return for their $30,000 deposit.

Other examples

The implications of this issue are far reaching throughout the entire protocol, and so we will not explain every vulnerability caused. At a high level, other vulnerabilities include:

- The `allocate` function of the `BondingCurve`, which allows PCV to be removed from the `BondingCurve` and into the rest of the protocol uses `readOracle` to calculate whether the value of the PCV held is large enough to allow the call to be successful. However the call will always fail until the amount of PCV held is `10 ** |feiDecimals - tokenDecimals|` larger than it should need to be. In the case of WBTC this would mean collecting billions of dollars to be able to remove the PCV from the contract – effectively locking it up until that happens.
- The `reweight` function of the `UniswapPCVController`, which rewards users for moving the FEI price back to the peg, incorrectly calculates whether a reweight should be allowed. The “distance to the peg” will be out by factors of 10, meaning that reweights will virtually always be possible.
- Additionally, when performing the reweight, the `_rebase` and `_reverseReweight` functions will be reweighting towards an incorrect peg price for FEI.

Consider updating the `readOracle` function to return the price in terms of token units, and not whole tokens. Alternatively, consider always adjusting the price returned from `readOracle` to account for the difference in token decimal amounts of the tokens in question. Additionally, consider updating your test suite to ensure that you are testing using a diverse set of tokens reflective of a mainnet environment.

Update: *Fixed in PR#69.*

[H01] Fei reweight and stabilization may not react to price swings

In the `TribeReserveStabilizer` contract, the `exchangeFei` function determines whether to exchange a users FEI for TRIBE by calling the `isFeiBelowThreshold` function. This function reads the current FEI price from an oracle, however this price is not ever updated by this contract, so a stale price is used to determine whether the exchange can occur.

Similarly, in the `UniswapPCVController`, the `reweight` function determines whether a reweight is allowed by immediately calling the `reweightEligible` function. This fetches the current uniswap price, and compares it to a price that it reads from an oracle. If the difference between the 2 prices is large enough, a reweight is permitted to occur. However the `update` function of the oracle is not called prior to reading the price, meaning that stale oracle prices can be used without detection.

Consider always updating oracle prices before reading their values, to avoid making incorrect decisions in critical protocol operations.

Update: *Partially fixed in PR#70. The* `reweight` *issue outlined above has been fixed, however* `exchangeFei` *issue has not been fixed. The Fei Protocol team stated:*

> Did not add for TribeReserveStabilizer because feiOracle will be a chainlink oracle without need for update

[H02] Introduction of additional BondingCurves creates period of volatility

The `BondingCurve` contract enables use of generic bonding curves for any ERC20 token, in addition to the ETH bonding curve already being used in the protocol. The `BondingCurve` mints new FEI tokens in exchange for a specified ERC20 `token` at a discounted price, until a `scale` number of FEI have been minted. Once the curve is at scale, the price will increase and maintain a `buffer` price.

The Fei team have stated that they intend for `scale` to be 1m-100m, and the discount to be 0-5%. Assuming the most extreme of these parameters, 100M FEI will be available to purchase for $95M of token. This large incentivization will likely lead to bots rapidly purchasing discounted FEI and immediately selling it for a profit on various exchanges. This means that every time a new

Uniswap pools.

Take for example a new DAI bonding curve, with a 5% discount, and a scale of at least 5M. Additionally assume that the ETH/FEI and ETH/DAI UniswapV2 pools have the same reserves and prices as at the time of writing. A user can:

- Flash-borrow $4.8M in DAI
- Buy $5.053M FEI from the `BondingCurve`
- Sell this FEI for 2326 ETH using the ETH/FEI UniswapV2 pool
- Sell this ETH for $4.93M DAI using the ETH/DAI UniswapV2 pool
- Pay back the flash loan and keep $137,000 as profit

While $137k of profit isn't as large as many of the attacks seen in the ecosystem, this method can be repeated across many different exchanges and pools to make considerably more money when scale is as large as 100M. Additionally with bots performing the trades, this will lead to the price of FEI crashing immediately after each new `BondingCurve` contract is deployed, and day to day users being disadvantaged by their stablecoin holding crashing in price. This crash in price will ultimately also lead to increased use of, and draining of PCV from, the various ReserveStabilizer contracts.

Consider removing the offering of discounted FEI for new tokens, to remove the added volatility of the price of FEI. Alternatively, consider significantly reducing the parameters of 100M and 5% so that the effect of launching a new `BondingCurve` contract will not be so large.

Update: *Not fixed. The Fei Protocol team states:*

> This is expected behavior. The parameters used will be conservative and account for risks of volatility in FEI.

[H03] Inconsistent use of oracles

Throughout the protocol, oracles are relied upon to keep FEI stable, calculate payouts to users, and judge whether actions are eligible to be carried out. In addition to the vulnerabilities caused by

Oracle updates

The functions `IOracle.update` and `OracleRef.updateOracle` return a boolean. However the definition of what this boolean signals is ambiguous. The `UniswapOracle` contract defines it as a signal whether an update was needed – `false` is not a failure, but instead just a signal that the oracle didn't need to be updated. Whereas the `OracleRef` contract defines it as a signal of whether the update was *effective* – here `false` is the signal of a failure to update the price.

This has lead to inconsistent handling of this boolean value: The `PCVSwapperUniswap.swap` function reverts if `false` is returned, whereas the remainder of the codebase ignores the return value entirely. If the definition found within `OracleRef` is correct, functions throughout the codebase are not reverting when the update fails, which they should be.

Use of `isOutdated`

All oracles of type `IOracle` implement a function `isOutdated` which signals whether or not the current oracle price is outdated. This function is utilized in `PCVSwapperUniswap`, however the rest of the codebase ignores the function, and never checks whether an oracle needs to be updated.

Inverting oracle prices

Oracles return a price of a currency X, in terms of a currency Y, which can then be inverted if what is desired is the price of Y in terms of X. However the possibilities of inverting token prices is inconsistent throughout the codebase. The `PCVSwapperUniswap` has a boolean flag `invertOraclePrice` which signals whether the oracle being used needs to be inverted or not, whereas the `ReserveStabilizer` always inverts the price, and the remainder of the codebase never inverts the price.

Consider updating the codebase to remove the above inconsistencies and vulnerabilities, and updating all comments and documentation to reflect the intended use of all functions and booleans.

throughout the codebase - the function is now not used at all. Moreover, given that the purpose of `isOutdated` *is to flag whether an oracle price is stale, reading oracle prices without first calling* `isOutdated` *could lead to incorrect prices being used throughout the protocol to determine critical conditions.*

Medium severity

[M01] Errors and omissions in events throughout codebase

Throughout the codebase, events are used to signify when changes are made to the contracts. However many events lack indexed parameters, are missing important parameters, or emit erroneous parameters. Additionally collisions between events exist, and some sensitive actions are lacking events altogether.

Events emitting erroneous parameters include:

- The constructor of `PCVDripController` erroneously emits the `_incentiveAmount` in the `DripAmountUpdate` event. It should instead emit the `_dripAmount`.
- The `dripAmount` emitted in the `Dripped` event of the `PCVDripController` can be the incorrect amount. The is due to the fact that the amount withdrawn from uniswap can differ from the amount requested. The actual amount dripped is equal to the `amountWithdrawn` in the `withdraw` function of `UniswapPCVDeposit`.

Events missing important parameters include:

- `UpdateReceivingAddress` should include the previous variable value.
- The 4 events defined in `PCVSwapperUniswap` should include the previous variable value.
- `TimerReset` should include the previous variable value.

Events lacking indexed parameters include:

- All events in `Timed`.
- All events in `Incentivized`.
- All events in `PCVSplitter`.

- `FeiPriceThresholdUpdate` in `ITribeReserveStabilizer`.
- `DripAmountUpdate` in `IPCVDripController`.
- `ScaleUpdate`, `BufferUpdate`, `DiscountUpdate`, and `Reset` in `IBondingCurve`.

Sensitive actions that are lacking events include, but are not limited to:

- The relevant "update" events in the constructors of `BondingCurve`, `PCVSwapperUniswap`, and `TribeReserveStabilizer`.

Additionally, some contracts have events with identical names. For example the `WithdrawERC20` event in `RatioPCVController` and `IPCVDeposit`. A call to `RatioPCVController.withdrawRatioERC20` emits 2 different events with different parameters, but both named `WithdrawERC20`. Another occurrence is the `Timed` contract and the `IUniswapOracle` interface both define a `DurationUpdate` event. This may cause confusion for off-chain clients and users alike.

Finally, the `ReweightIncentiveUpdate` event is defined but never used.

Consider making all of the above changes to enable off-chain clients to correctly track changes to the Fei Protocol.

Update: *Partially fixed in PR#104. The following items were not fixed:*

- *The `dripAmount` of the `Dripped` event remains potentially inaccurate.*
- *2 of the 4 events in `PCVSwapperUniswap` do not emit the previous variable value.*
- *`TimerReset` does not emit the previous value.*
- *No indexing has been added to the list of events lacking indexed parameters provided.*

[M02] `UniswapPCVDeposit` can `burn` PCV FEI without Burner role

The external `deposit` and `withdraw` functions of the `UniswapPCVDeposit` contract each respectively have underlying calls to the `_burnFeiHeld` function of the `CoreRef` contract. This `_burnFeiHeld` function has no modifier or check that the caller has the Burner role.

callable by anyone, it will only burn PCV addressmaking after adding liquidity to the Uniswap pool. However the `withdraw` function will burn non-trivial PCV FEI resulting from its underlying `removeLiquidity` call to the Uniswap pool. While the `withdraw` function is modified by `onlyPCVController`, it can be triggered by anyone by calling the `drip` function of the `PCVDripController` contract. This means, in the case the Burner role of `UniswapPCVDeposit` is revoked as an emergency incident response, the safeguard can be circumvented.

Consider refactoring the `_burnFeiHeld` function to ensure the caller has the Burner role.

Update: *Not fixed. The Fei Protocol team state:*

> Any address can burn its own FEI, this is expected behavior

[M03] Lack of input validation

Throughout this codebase we found there to be an overall lack of input validation. The functions lacking input validation are either modified by the `onlyGovernance` modifier or are `constructor`s. However simple human error in entering these values, by perhaps entering too many or too few 0s, can have far reaching negative consequences.

Some points where a lapse in input validation could be particularly problematic include:

- There is no check in the callstack of the `setAllocation` function of the `BondingCurve` contract to ensure that the `token` featured in the `BondingCurve` is the `token` handled by the `PCVDeposit`. In the case this mismatch would occur, the `PCVController` would have to manually reallocate these stray tokens using the `withdrawERC20` function.

- The `PCVDripController` contract drips tokens from one `PCVDeposit` contract to another, but it never validates that the token each of them handle are the same. This means it could drip a token into a deposit contract that handles a different token. This mismatch could affect the accounting dictating the logic of the `drip` function. In particular, the check of `dripEligible` may wind up considering the balance of the wrong token.

Examples of `onlyGovernor` modified functions lacking input validation are:

- The `setFeiOracle` function of the `TribeReserveStabilizer` contract doesn't check `newFeiOracle` is not the zero-address.

- The `setDuration` function of the `UniswapPCVController` contract does not validation the `_duration` is non-zero or within sensible bounds.

The functions that could benefit from input validation in this codebase are numerous and there are many more than are listed here.

Consider implementing programmatic safeguards validating input parameters to ensure all function calls and contract constructions would "fail early and loudly" on erroneous inputs. This is needed especially in the case of functions or contracts vetted by governance, where subtle bugs in parameters that pass the governance process can have far reaching impacts on a system.

Update: *Partially fixed in PR#75. The first 2 points outlined in this issue were not addressed, however the remaining issues were fixed.*

[M04] `ReserveStabilizer` performs unsafe token transfer

The `exchangeFei` function of the `ReserveStabilizer` contract burns a user's FEI and in exchange transfers them tokens at a discounted rate. However, the boolean return value of the token transfer performed is not checked. This means that ERC20-compatible tokens that return false on failure, such as ZRX, will not cause the call to revert. In turn the user will receive no tokens for their FEI.

Consider updating the token transfer to instead use OpenZeppelin's SafeERC20, which will handle all edge cases of token transfers for you.

Update: *Fixed in PR#71.*

[M05] Incorrect use of Time-Weighted Average Prices

The oracle interface is designed such that the protocol can use any combination of oracles from different protocols, but currently only those from Chainlink, and Uniswap have been integrated – with the live protocol solely using Uniswap at launch in January, and now solely Chainlink. However, the way that the protocol inverts the price read from `IOracle` is not safe when the price is a Time-Weighted Average Price (TWAP) calculated with an arithmetic mean, as is the case with Uniswap V2.

To protect against price manipulation, UniswapV2 uses Time-Weighted Average Prices (TWAPs). These TWAPs enable contracts to fetch an average price over a given time interval, instead of using the asset spot price at the current instant. This means that prices are significantly harder to manipulate, which is a great safety feature for protocols using these prices. However, while the spot price of `token0/token1` can be inverted to get the spot price of `token1/token0`, the same is not true of TWAPs: the TWAP of `token0/token1` *cannot be inverted* to get the TWAP of `token1/token0`. The more volatile the price of `token0/token1`, the more extreme this result is.

For example, assume we have tokens `ABC` and `DEF`, where the price of `ABC/DEF` at time 1 is 100, and the price at time 5 is 700. With some simplification of scaling, the cumulative prices tracked in UniswapV2 would be:

```
cumulativeABC: (1*100)+(4*700) = 2900
cumulativeDEF: (1*1/100)+(4*1/700) = 11/700
```

The average prices over the window from 0 to 5 are therefore calculated as:

```
averagePriceABC: 2900/5 = 580
averagePriceDEF: 11/700/5 = 0.00314 (approx.)
```

If instead of using the `DEF` price of `0.00314`, the inverted `ABC` price is used, this is a price of `1/580 = 0.00172`. This is approximately 54% of the correct price.

The codebase twice uses the `IOracle.invert` function to calculate the reciprocal of a price like this – without knowledge of whether the oracle is using Chainlink spot prices, Uniswap TWAPs,

incorrectly by potentially large percentages.

Consider updating the code to ensure that Uniswap TWAPs are never inverted using a reciprocal, and instead the inverse accumulator is used to correctly calculate the inverse TWAP.

Update: *Partially fixed in PR#69. The updated code makes it possible to use oracles correctly and safely, and is a complete fix as long as one condition is never broken:* ***If the*** `oracle` ***or*** `backupOracle` ***in*** `OracleRef` ***returns a Uniswap TWAP as the price, the*** `doInvert` ***variable should never be set to*** `true`***.*** *If this condition is broken, the prices used throughout the codebase will be incorrect, and could lead to vulnerabilities.*

Low severity

[L01] Duplicate definitions

The `WETH` `address` is set as a `public` `immutable` state variable in:

- `PCVSwapperUniswap`
- `EthReserveStabilizer`

The `uint256` `constant` `BASIS_POINTS_GRANULARITY` is defined in various parts of the protocol, all having the same value of `10_000`:

- `BondingCurve`
- `UniswapPCVDeposit`
- `PCVSwapperUniswap`
- `RatioPCVController`
- `UniswapPCVController`
- `ReserveStabilizer`

While these definitions were made with consistent values across the codebase, the multiple definitions could introduce inconsistencies in future updates to the protocol

Consider defining values shared across the protocol once in a common library.

Update: *Not fixed.*

using ERC20 tokens. Due to the fact that `EthBondingCurve` inherits `BondingCurve`, and requires its `purchase` function to be payable, `BondingCurve`'s `purchase` is also payable, despite not handling the case where ETH is sent with a call. Furthermore, once ETH is in the `BondingCurve`, there is no way to withdraw the ETH again.

Consider requiring that the `msg.value` of a call to `BondingCurve.purchase` is 0 to prevent ETH getting stuck inside the contract.

Update: *Fixed in PR#72.*

[L03] Using old Solidity version

Throughout the contract, Solidity `^0.8.0` is used. However, at the time of writing there are 2 known bugs in version 0.8.0. Consider upgrading your contracts to use the latest version of Solidity, 0.8.7.

Update: *Fixed in PR#73.*

[L04] Role definition differs from implementation

The "Tribe Minter" role is defined in the Fei protocol to only be held by a single contract. This role, as the name suggests, controls the ability of an entity to mint TRIBE. Currently this role is held by the `Timelock` contract but it is planned that this "Tribe Minter" role will be transitioned to the `TribeReserveStabilizer` contract. The `TribeReserveStabilizer` needs the "Tribe Minter" role since it mints TRIBE to exchange for FEI in its `exchangeFei` function.

However the `TribeReserveStabilizer` also has as one of its methods a `mint` function, modified by the `onlyGovernor` modifier, that makes an internal call to mint TRIBE. This means that the `Timelock` contract retains its ability to mint TRIBE by way of this `onlyGovernor` modified `mint` function despite passing this distinct role to the `TribeReserveStabilizer`. In other words, the `Timelock` contract effectively shares the "Tribe Minter" role with the `TribeReserveStabilizer` despite the role being documented to be held by a single contract. This discrepancy can be misleading to users in understanding the security model of the Fei protocol.

Update: *The Fei Protocol team state:*

> We will update the documentation.

[L05] Uninitialized global variable

The `OracleRef` contract relies on 2 oracle addresses – the main `oracle`, and a `backupOracle`. However, only the `oracle` can be initialized upon construction of the contract – the `backupOracle` remains uninitialized even if the creator wants the contract to have a backup oracle.

Consider adding a parameter to the constructor of `OracleRef` to allow the `backupOracle` to be initialized on creation of the contract.

Update: *Fixed in PR#67.*

[L06] Variable shadowing in ReserveStabilizer

The `ReserveStabilizer` contract has a global variable `token`, which stores the address of the token exchanged on this stabilizer. The `ReserveStabilizer.withdrawERC20` function additionally defines a local variable named `token`, shadowing the global variable definition. Creating shadowed variable names can lead to bugs where the writer intends to use the global variable, but the compiler uses the local shadow variable instead. Additionally it makes the code harder to read, and confusing to understand.

Consider renaming the local `token` variable within `withdrawERC20`.

Update: *Fixed in PR#74.*

Notes & Additional Information

[N01] `_getAmountToPeg` lacks comments

The `_getAmountToPeg` function of the `UniswapPCVController` contract performs a key calculation in the reweight algorithm. If the Uniswap price of Fei in relation to the other token is

The `_reverseReweight` relies on calculations from the `_getAmountToPeg` function involving a fairly simple formula. The derivation of this formula is not immediately straight-forward but appears in the Fei protocol's documentation. Since this derivation appears in external documentation but not commented in the code, it is easy for readers to become confused and miss the connection.

To aid in understanding and readability, consider documenting as comments in code derivations of all mathematical formulae.

Update: *Fixed in PR#76.*

[N02] Interfaces missing functions

Many of the contracts in the codebase have a matching interface which defines the function signatures for the contract. However some of the interfaces are missing functions which are defined in the respective implementing contract. Examples of this are:

- `IUniswapPCVDeposit` is missing `setPair`.
- `IPCVSwapper` is missing `wrapETH`, `setMaximumSlippage`, `setMaxSpentPerSwap`, `setSwapFrequency`, and `setInvertOraclePrice`.
- `IReserveStabilizer` is missing `token`.

Consider updating these interfaces to define the missing function signatures.

Additionally, `ITribeReserveStabilizer` is missing all of the functions defined in `IReserveStabilizer`. Consider having `ITribeReserveStabilizer` inherit `IReserveStabilizer`.

Update: *Not fixed.*

[N03] Missing Natspec

While the majority of the codebase is well-commented, with Natspec used for most functions, there are some places that Natspec is missing or incomplete. Some examples of this are:

- The `_token` parameter of the `ReserveStabilizer`'s constructor.

`PCVSwapperUniswap.setInvertOraclePrice`.

- The return parameters throughout `PCVSwapperUniswap`.
- The parameters throughout `RatioPCVController`.
- The documented order of the constructor parameters in `BondingCurve` is inconsistent with their order in the constructor definition.

Additionally, some of the existent Natspec contains erroneous information. Examples of this include:

- The `BondingCurve` constructor states its parameter will be a `UniswapOracle`, however this is not the case.
- The Natspec documenting `PCVSwapperUniswap`'s `_getDecimalNormalizer`, `_getMinimumAcceptableAmountOut`, `_getExpectedAmountOut`, and `_getExpectedAmountIn` references 4 named external functions that do not exist in the codebase.
- The Natspec for `_getBondingCurvePriceMultiplier` states that it calculates a multiplier "at the current totalPurchased relative to Scale", however neither `totalPurchased` nor `scale` factor into the calculation.

Consider updating the above examples and checking that complete Natspec exists for all functions throughout the codebase.

Update: *Partially fixed in PR#79. The NatSpec of the following functions was not fixed:* `PCVSwapperUniswap.setInvertOraclePrice`*, and* `PCVSwapperUniswap._getDecimalNormalizer`*.*

[N04] Renaming suggestions

Good naming is one of the keys for readable code, and to make the intention of the code clear for future changes. There are some names in the code that make it confusing or hard to understand.

Consider the following suggestions:

In the `UniswapPCVController` contract:

In the `BondingCurve` contract:

- `adjustedAmount` to `feiValueOfAmountIn`.

Update: *Fixed in PR#77.*

[N05] Inconsistent style

There are various occurrences of style inconsistency within this codebase:

- The constant definition to value `10000` is inconsistent with all other definitions having value "10,000" using the "underscore" notation `10_000`.
- Unnecessary indentations on L248 of `BondingCurve.sol` and throughout `PCVSwapperUniswap.sol`.
- Inconsistent use of underscores "`_`" in parameters where they are not used in `TribeReserveStabilizer` contract, they are used occasionally in `UniswapPCVController` and `BondingCurve` contracts, and are used everywhere in `RatioPCVController` contract.

Taking into consideration how much value a consistent coding style adds to the project's readability, enforcing a standard coding style is recommended.

Update: *Partially fixed in PR#77. However both the indentations throughout `PCVSwapperUniswap.sol`, and the use of underscores in parameters throughout the codebase remain inconsistent.*

[N06] Typos

The codebase contains the following typos:

- `innacurate` should be `inaccurate`.
- `oldReweigtMinDistanceBPs` should be `oldReweightMinDistanceBPs`.
- `drip ETH to target` should be `drip PCV to target`.
- `ETH PCV Dripper` should be `PCV Dripper`.

Consider correcting these typos to improve code readability.

In the `PCVSplitter` contract, the `checkAllocation` function returns a boolean. However the function always reverts on failure, and the only call to this function does not check its return value.

Similarly, the `_getUniswapPrice` function in `UniswapPCVController` returns 3 parameters, but only the first of these is ever used.

Consider removing the unused return value from the function to simplify the codebase.

Update: *Fixed in PR#78.*

[N08] Unnecessary imports

Consider removing the following import statements, as they are never used in their corresponding contracts:

- `import "@openzeppelin/contracts/utils/Address.sol";` in `CoreRef.sol`.
- `import "@openzeppelin/contracts/utils/math/Math.sol";` in `BondingCurve.sol`.
- `import "@openzeppelin/contracts/token/ERC20/IERC20.sol";` in `PCVSplitter.sol`.
- `import "@openzeppelin/contracts/token/ERC20/IERC20.sol";` in `IUniswapPCVDeposit.sol`.
- `import "@openzeppelin/contracts/utils/math/Math.sol";` in `UniswapPCVController.sol`.

Update: *Fixed in PR#77.*

[N09] Using ERC20 not IERC20

In the `PCVSwapperUniswap` contract, `ERC20` is imported to allow the contract to interact with a variety of tokens. Throughout the rest of the codebase, as is customary, the interface `IERC20` is instead used to interact with tokens.

Update: *Fixed in PR#69.* `IERC20MetaData` *was used instead of* `IERC20`*, as the metadata functions were required to fix another issue.*

Conclusions

1 critical and 3 high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin

© Zeppelin Group Limited 2023