PUBLIC

Code Assessment

of the Frax Upgrade Smart Contracts

July 17, 2025

Produced for

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Frax Team,

Thank you for trusting us to help Frax with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Frax Upgrade according to Scope to support you in forming an opinion on their security risks.

Frax implements upgrades for the Frax infrastructure which include the upgrade of LFRAX and LFRAX_FXB, the implementation of a beacon factory for future FXBs, the upgrade of the sfrxUSD and the respective oracles on Fraxtal as well as the implementation of an sfrxUSD minter and redeemer on Fraxtal.

The most critical subjects covered in our audit are the impact of the upgrades, e.g., in the storage layout, the correctness of the calculations, and accounting within the system. Several low issues were uncovered. More specifically, the system has been implemented to be used with 18 decimal tokens only but this assumption is not enforced (Unverified assumptions can break the code), some minor issues arise from rounding errors (Rounding errors break some ERC-4626 calculations) and from the way the oracles update (An outdated price is used in computations). All the issues have been addressed or acknowledged by the team.

The general subjects covered are compliance with various ERCs, gas efficiency, testing, documentation, and specification. The comments in some areas of the code could improve. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	10
• Code Corrected	6
• Specification Changed	1
• Risk Accepted	3

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Frax Upgrade repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

Frax Bonds

V	Date	Commit Hash	Note
1	23 June 2025	fe4caf06dc83debf62de40d7b3f2834ed778f5fb	Initial Version
2	14 July 2025	4ef235e68c8277ad3dbc676762fffe663c9896fc	Fixes

Staked Frax

V	Date	Commit Hash	Note
1	23 June 2025	1e0f20629cabc4a565b90066146878a2ccd71735	Initial Version
2	14 July 2025	5e69987ba2af1514baf8708b2d9582a0c45bd048	Fixes

For the solidity smart contracts, the compiler version `0.8.19` for Frax bonds and `0.8.21` for the staked frxUSD was chosen.

We considered the following contracts of the Frax Bonds repo:

```
src/contracts/

    FXB_LFRAX.sol
    FraxBeacon.sol
    FraxBeaconProxy.sol
    FraxUpgradeableProxy.sol
    FXB.sol
    FXBFactory.sol
```

As part of the scope we also considered the following scripts:

```
src/script/LFRAX_FXB_Upgrade/

    00_DeployImpls.s.sol
    01_HandleNewLFRAX.s.sol
    02_UpgradeFXBs.s.sol
```

We considered the following contract from the Staked Frax repo:

```
src/contracts/

    StakedFrxUSD2.sol
    LinearRewardsQuasiErc4626.sol
```

```
FraxtalERC4626MintRedeemer.sol
SfrxUsd2OracleImplementation.sol
```

2.1.1 Excluded from scope

All contracts not mentioned in scope are considered out of scope. In particular third-party libraries used within the scope are assumed to function properly. The Fraxtal L2 where some contracts are deployed is assumed to implement similar semantics to EVM and its sequencer is fully trusted. The financial modeling of the system is beyond the scope of this review. For the compilation of the smart contracts, we assume unmodified versions of solidity will be used.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Frax offers some upgrades to the Frax infrastructure, both on Ethereum and Fraxtal. These include 1) the LFRAX contract upgrade, 2) the FXB upgrade, 3) the implementation of beacon for FXBs, 4) the upgrade of sfrxUSD to version 2, 5) the upgrade of the sfrxUSD Oracle to version 2, and 6) the implementation of the FraxtalERC4626 minter and redeemer.

2.2.1 LFRAX contract upgrade

LFRAX is the legacy Frax Dollar contract. The contract is being phased out in favor of frxUSD (also called FRAX). The contract was upgraded to be an `OptimismMintablePermitERC20` contract.

2.2.2 FXB Upgrade

FraxBonds (FXBs) are zero-coupon bonds deployed on Fraxtal. Users buy FXB at a discount and receive the underlying at maturity. The contract allows users to deposit the underlying in exchange for an ERC20 token representing it. The already deployed FraxBonds are upgraded to replace the FrxUSD backing them with LFRAX. The liquidity of the FrxUSD will be removed upon the upgrade. This way, Frax can use the FrxUSD, whose circulation is subject to restrictions in other DeFi protocols. On the other hand, Frax is in full control of the LFRAX circulation. Therefore, at maturity, Frax will mint LFRAX tokens backing the circulation of the FXB token. During the upgrade, the version field of the FXB for EIP712 will change to 1.2.0 invalidating pending permits.

2.2.3 FXBv2 Beacon

For the upcoming FXBs the deployment process is going to be different as they'll be deployed as beacon proxies:

- FraxUpgradeableProxy: the proxy to FXBFactory implementation.
- FXBFactory: it allows its owner to create new bonds (`createFxbContract`).
- FraxBeaconProxies: the proxies created by the factory.
- FraxBeacon: it stores the current implementation of the FXBs.
- FXB: the current implementation. It's essentially an ERC20Upgradeable with permit.

2.2.4 Staked frxUSD Version 2

Staked FrxUSD (SfrxUSD) is an ERC4626 contract deployed on Ethereum that allows users to stake their FrxUSD for yield. SfrxUSD will be updated to version 2. The upgrade disables the ERC4626 functionality of the contract. Only designated minters can arbitrarily mint or burn assets. The minters are added or removed by the Frax multisig. Upon initialization of the new version, the FrxUSD tokens will be burnt. In its new version, the contract calculates the price of the SfrxUSD using a continuous compounding formula parametrized by the price increase per second. Anyone can checkpoint the price by calling `sync()`. The multisig can update the parameters of the formula and arbitrarily set the price.

The price evolution of the SfrxUSD is going to be tracked by the updated implementations of oracles on Fraxtal. The oracles similarly calculate the price as sfrxUSD.

2.2.5 FraxtalERC4626MintRedeemer

This contract allows the minting and redemption of bridged SfrxUSD on Fraxtal in exchange for FrxUSD. Even though not an ERC4626 per se, the contract implements an ERC4626-like interface. Users can call `mint()`/`deposit()` to receive SfrxUSD or `burn()`/`redeem()` to receive pre-deposited FrxUSD.

2.3 Trust Model

We infer the following trust model:

- Frax Admin: The role is **fully trusted**. More specifically:
    - They have full control of all the contract implementations, which they can upgrade at will. Should the admin set a malicious implementation users' funds can be at risk. This includes users of protocols that integrate with Frax.
    - They are trusted to deposit the required balance of LFRAX to the FXB_LFRAX for users to redeem their FXBs.
    - They can parameterize the FXBs. The parametrization is assumed to always be reasonable.
    - Through their multisig/timelock, they can specify the minters for the sfrxUSD.
    - They can set the oracle addresses of the FraxtalERC4626MintRedeemer. The oracles should use the same precision and denomination asset.
- SfrxUSD Minters: They are **fully trusted**, they can arbitrarily burn and mint sfrxUSD for any address.
- Oracle Price setters: The role is **fully trusted**. They can arbitrarily set the price parameters that determine the price evolution of sfrxUSD. Should the parametrization be wrong, systems that make use of sfrxUSD can be in danger.
- The system is assumed to be solvent when it's supposed to be. For example, users should be able to claim the underlying of their FXB or redeem their sfrxUSD.
- A big part of the system is deployed on the Fraxtal L2. The admins of the rollup have full control over the sequencing, safety, and security of the system. They can arbitrarily change the behavior of the system via hard forks.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

5.1 Breaking ERC-4626

Correctness Low Version 1 Risk Accepted

CS-FRUP-002

In `LinearRewardsQuasiErc4626`, the 4 ERC-4626 preview<X> functions (<X> is one of the operations: deposit, withdraw, redeem, and mint) always return 0. This breaks the standard that states:

> MUST NOT account for <X> limits like those returned from max<X> and should always act as though the <X> would be accepted, regardless if the user has enough shares, etc.
>
> MUST NOT revert due to vault-specific user/global limits. MAY revert due to other conditions that would also cause withdraw to revert.

Returning a 0 essentially implies a global limitation. Integrating protocols might malfunction due to that.

Risk accepted: The issue will not be fixed as it is unlikely to arise in practice.

5.2 Rounding Errors Break Some ERC-4626 Calculations

Correctness Low Version 1 Risk Accepted

CS-FRUP-005

In `FraxtalERC4626MintRedeemer`, rounding errors can lead to incorrect calculations breaking the ERC-4626 standard.

1. Calling `previewRedeem` with `maxRedeem` can return a value that is greater than the actual redeemable amount (the balance of the contract of the underlying token). This error is due to rounding in the `maxRedeem` calculation. Therefore, a maximum redemption could fail.

To illustrate this, consider the folowing scenario where:

- The fees are 0 and both the shares and the underlying use 2 decimals. The price of the shares is 3.00 (`Vprice = 300`) and the assets is 1.00 (`Uprice = 100`). The effect should be the same with 18 decimals but the calculations are less comprehensible.
- A user holds 10 shares.
- The vault holds `underlying.balanceOf(this) = 2`
- The user wants to redeem all their shares:

```
maxRedeem
    previewWithdraw
        convertToShares ceil(2 * <Uprice> 100 / <Vprice> 300) = 1

 min(1, 10) = 1, ⇒ 1 shares should be taken from the user
```

- Then the user calls:

```
previewRedeem(1)
    floor(1* <Vprice> 300 / <Uprice> 100) = 3
```

- The call will fail as the vault only holds 2 tokens of the underlying; therefore, the withdrawal fails.

2. Calling `previewDeposit` with `maxDeposit` can return a value that is greater than the actual number of shares that can be transferred to the user (the balance of the contract of the shares). This error is due to rounding in the `maxDeposit` calculation. In particular, the underestimation of `previewDeposit()` is not enough to correct the overestimation of `maxDeposit()`. Therefore, a maximum deposit can fail. Note that for this to happen, the price of the shares must be greater than the price of the underlying token.

5.3 Token Name on the Proxy Can Mismatch the Name on the Implementation Contract.

Correctness **Low** **Version 1** **Risk Accepted**

CS-FRUP-007

`StakedFrxUSD2.initialize` writes the token's `name` to proxy storage. However, the implementation contract is a non-upgradable `ERC20`. its constructor already sets the `name` and immediately used that value to pre-compute the immutable variable `INITIAL_DOMAIN_SEPARATOR` (EIP-712 domain separator):

Because the domain separator is immutable, updating the `name` later via `initialize` has no effect on `INITIAL_DOMAIN_SEPARATOR`. Consequently, the proxy may expose one token name while the implementation's domain separator embeds a different name. Any off-chain logic that relies on the visible proxy name (e.g., signing or verifying EIP-712 messages) can therefore use a mismatched domain separator, causing signature verification to fail.

Risk accepted: Frax will not address the issue as it doesn't affect this instance but they know how to address the issue should the need arise.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	7

- An Outdated Price Is Used in Computations Code Corrected
- Missing Check for lastSync Code Corrected
- Oracle Time Tolerance Is Not Respected Specification Changed
- Price Initialization Mismatch Code Corrected
- The Initial Price Is Not Defined Correctly Code Corrected
- Underlying Price Model Ambiguity Code Corrected
- Unverified Assumptions Can Break the Code Code Corrected

Informational Findings	7

- Fee Can Be Set to Wrong Value Code Corrected
- Initializer Not Disabled Code Corrected
- Misleading Natspec Code Corrected
- Missing Events Code Corrected
- Some Events Could Be Indexed Code Corrected
- _previewPricePerShare Should Be Internal Code Corrected
- calcPPSIPSForGivenAPY Missing Input Validation Code Corrected

6.1 An Outdated Price Is Used in Computations

Security Low Version 1 Code Corrected

CS-FRUP-001

In `FraxtalERC4626MintRedeemer`, `_convertToShares` and `_convertToAssets` functions are using `getLatestUnderlyingPriceE18` and `getVaultTknPriceStoredE18` to compute the shares and assets, respectively.

However, the first gets a fresh price from the oracle, while the second uses the stored price, which can be older than the first one, and thus lead to inconsistencies in the computations.

This basically allows the user to choose the price they want to use, either the latest one or the stored one. Moreover, it gives users the opportunity to wrap other actions around a price shift.

Code corrected: The code was changed, both prices are now fetched in the same way and updated before any state-changing operation.

6.2 Missing Check for lastSync

Correctness | Low | Version 1 | Code Corrected

CS-FRUP-003

`SfrxUsd2OracleImplementation.setAllPricingParams` is missing a check to ensure `lastSync` is not in the future.

Code Corrected: The check was added.

6.3 Oracle Time Tolerance Is Not Respected

Correctness | Low | Version 1 | Specification Changed

CS-FRUP-004

In `FraxtalERC4626MintRedeemer` the variable `oracleTimeTolerance` is a value that defines how old the oracle price should be to consider it stale.

This value is also used to evaluate the staleness of the stored/cached token value. This leads to stale prices, with respect to the price oracle, to be accepted. Assume `oracleTimeTolerance` is set to 1 hour:

1. At `t0` `updateVaultTknOracle` is called, resulting in calling `getLatestVaultTknPriceE18`. Assume `getLatestVaultTknPriceE18` returns the price with `updatedAt = t0 - 1 hour`, which is acceptable. We then set the `lastVaultTknOracleRead` to `block.timestamp`.
2. At `t1 = t0 + 1 hour`, we call `getVaultTknPriceStoredE18`. This will return the cached price because `lastVaultTknOracleRead + oracleTimeTolerance = t0 + 1 hour`. However, we are actually getting the price from the oracle that was set at `t0 - 1 hour`, which is stale because it is older than `oracleTimeTolerance`.

Specification changed:

`lastVaultTknOracleRead` is now set to `block.number` when the oracle is read regardless of the `updatedAt` value. `oracleTimeTolerance` is now only used to check the staleness of the oracle price, not the cached value.

6.4 Price Initialization Mismatch

Security | Low | Version 1 | Code Corrected

CS-FRUP-023

In `FraxtalERC4626MintRedeemer.initialize`, the price oracles and the initial price are set. However, this price can mismatch the current oracle price, since it is set manually. This mismatch can be abused by a malicious user backrunning the initialization.

Code corrected: The price is now fetched from the oracle upon initialization.

6.5 The Initial Price Is Not Defined Correctly

Security | Low | Version 1 | Code Corrected

CS-FRUP-006

When initializing `SfrxUsd2OracleImplementation`, the initial price is not set.

If a user backruns the contract initialization and calls `latestRoundData`, they will get a price of `0` for the actual round, which is not correct.

Code Corrected:

The price is now set upon initialization.

6.6 Underlying Price Model Ambiguity

Correctness | Low | Version 1 | Code Corrected

CS-FRUP-008

`FraxtalERC4626MintRedeemer` mixes two incompatible approaches for valuing the underlying token:

1. Relative-price model: the price of the underlying is constant `1e18`. Only the vault token changes. This path is taken when `priceFeedUnderlying` is the zero address.
2. Dual-oracle model: both tokens have independent oracles. When `priceFeedUnderlying` is non-zero, the contract calls `getLatestUnderlyingPriceE18` and applies staleness checks.

Having both paths active introduces several inconsistencies:

- Asymmetric caching: the vault token price is cached in `vaultTknPrice` and gated by `lastVaultTknOracleRead`. The underlying price is always fetched live. Every conversion therefore, combines a stored vault price with a live underlying price.
- One-sided initialization: `initialize` receives `_initialVaultTknPrice` but no initial underlying price.
- No update function for the underlying oracle: the contract exposes `updateVaultTknOracle` but nothing equivalent for the underlying. Timing differences between Oracle updates can lead to arbitrage opportunities.

Code corrected:

The two tokens are now treated consistently.

6.7 Unverified Assumptions Can Break the Code

Correctness | Low | Version 1 | Code Corrected

CS-FRUP-009

In multiple places, the code assumes that the underlying token has 18 decimals. This is not guaranteed and can lead to incorrect calculations if the underlying token has a different number of decimals:

1. `LinearRewardsQuasiErc4626._previewTotalAssets` uses the magic number `1e18` to scale back to the correct precision, using the assumption that the underlying token has 18 decimals.
2. `LinearRewardsQuasiErc4626._previewPricePerShare` uses the same assumption, some part of the computation would not work with other precision, notably `_exponentUD60_18` would take an incorrect value. The same applies to `SfrxUsd2OracleImplementation._previewPricePerShare`.
3. `FraxtalERC4626MintRedeemer._pricePerShare` assumes that one share has 18 decimals. However, since this is nowhere enforced, should the shares have different decimals the calculation will be wrong.

Code corrected:

`LinearRewardsQuasiErc4626` and `FraxtalERC4626MintRedeemer` now ensure that the underlying token has 18 decimals. Note that we assume that any contract that inherits from `LinearRewardsQuasiErc4626` will use the same token, ensuring the same precision.

6.8 Fee Can Be Set to Wrong Value

Informational **Version 1** **Code Corrected**

CS-FRUP-012

In `FraxtalERC4626MintRedeemer.setMintRedeemFee`, the fee value is checked to be a fraction of the underlying. This check is missing in the `initialize` function.

Code corrected: The fee is now checked in the `initialize` function to be less than `1e18`.

6.9 Initializer Not Disabled

Informational **Version 1** **Code Corrected**

CS-FRUP-014

FXB is missing the call to `_disableInitializers()` in the constructor.

Code corrected: The constructor was updated to include the call to *_disableInitializers()*.

6.10 Misleading Natspec

Informational **Version 1** **Code Corrected**

CS-FRUP-015

Some natspec comments are misleading or incorrect. Consider updating the following:

1. FXB_LFRAX has a misleading natspec comment `EIP721` when it should be `EIP712`.
2. `SfrxUsd2OracleImplementation` has a misleading category name `Events` for the errors.

Code corrected: Natspec comments were updated.

6.11 Missing Events

Informational | Version 1 | Code Corrected

CS-FRUP-016

Certain state-changing operations are not reflected in contract events, reducing observability for off-chain indexers and front-end integrations.

Declared but not emitted:

1. `StakedFrxUSD2.initialize` doesn't emit events when some state variables are set, even though such events are emitted by the respective setters.
2. `FraxtalERC4626MintRedeemer.initialize` misses the event when the owner is set.

Useful events not declared:

1. `FraxtalERC4626MintRedeemer` is missing an event when `fee` is set.
2. `FraxtalERC4626MintRedeemer` is missing events when oracles are set.

Code corrected:

The new events were added to the contracts. Note that still no events are emitted in the `initialize` functions.

6.12 Some Events Could Be Indexed

Informational | Version 1 | Code Corrected

CS-FRUP-018

Some events are not indexed, which makes them less efficient to search for. Consider indexing the following events:

1. `FraxBeacon.FraxBeaconImplementationUpdated`
2. `FXBFactory.BondCreated`

Code corrected: The events listed are now indexed.

6.13 `_previewPricePerShare` Should Be Internal

Informational | Version 1 | Code Corrected

CS-FRUP-021

`LinearRewardsQuasiErc4626._previewPricePerShare` and `SfrxUsd2OracleImplementation._previewPricePerShare` are declared `public` even though they serve only as internal helpers. External callers should use `previewPricePerShareFuture` for share-price estimates. The naming violates the Solidity convention for public functions.

Code Corrected: The functions have been changed to `internal` visibility.

6.14 `calcPPSIPSForGivenAPY` Missing Input Validation

Informational | Version 1 | Code Corrected

CS-FRUP-022

`LinearRewardsQuasiErc4626.calcPPSIPSForGivenAPY` implicitly assumes that `_apyE18` is at least `1e18`. If a smaller value is supplied, `ln(wrap(_apyE18))` will revert without an explicit guard.

Code corrected: An explicit check was added.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Code Simplification by Inheritance

Informational | Version 1 | Acknowledged

CS-FRUP-010

`FXB_LFRAX` re-implements the `EIP-712` domain separator logic and helpers locally instead of inheriting `EIP712Upgradeable` from OpenZeppelin. This would avoid unnecessary lines in the codebase, making audits and future maintenance easier.

Acknowledged: Frax said that this is due to legacy reasons and that they will not address it.

7.2 Far in the Future, Times Result in Computation Errors

Informational | Version 1 | Acknowledged

CS-FRUP-011

The `LinearRewardsQuasiErc4626._previewPricePerShare` will lose precision when the time is unrealistically far in the future, resulting in incorrect results. This can also cause a revert.

The same applies to `SfrxUsd2OracleImplementation._previewPricePerShare`, and to helpers calling it.

Comparing a normal update process on an hourly basis with a one-shot update, we can see that the accumulation error due to underapproximation in the update will lead to the following deviation:

- 4.3e10 wei over 10 years
- 4.1e16 wei over 20 years
- 2.9e22 wei over 30 years

7.3 Gas Optimizations

Informational | Version 1 | Code Partially Corrected

CS-FRUP-013

The following changes may reduce deployment or runtime gas costs without affecting functionality:

1. `FXB_LFRAX.initialize` uses a parameter to set the factory address, but according to the documentation this is always the `msg.sender`.
2. `FXBFactory.InvalidMonthNumber` is not used.
3. `FXB.totalFxbMinted` and `FXB.totalFxbRedeemed` are not used, only set. Also, the total supply is available via the `totalSupply()` function.
4. `FXB.burn` doesn't fail early when there isn't enough underlying.

5. `FXB_LFRAX.FXB_LFRAX` is not used, only set.
6. `StakedFrxUSD2.initialize` calls `sync` which does not affect the state.
7. `StakedFrxUSD2.removeMinter` can be optimized out of the loop, letting the caller handle the selection.
8. `LinearRewardsQuasiErc4626.calcPPSIPSForGivenAPY` can be pure.
9. `LinearRewardsQuasiErc4626.calcPPSIPSForGivenAPY` computes `ln(wrap(1e18))` which is always 0.
10. `LinearRewardsQuasiErc4626.calcPPSIPSForGivenAPY` computes `convert(ONE_YEAR)` which can be precomputed.
11. `LinearRewardsQuasiErc4626._previewPricePerShare` redundantly converts `pricePerShareIncPerSecond * _elapsedTime` and `UNDERLYING_PRECISION`. These values are in 18-decimal precision. Note that the current implementation is correct as the extra decimals cancel out.
12. `LinearRewardsQuasiErc4626.totalAssets` calls `previewPPSAndTotalAssets` when only the total assets are needed.
13. `LinearRewardsQuasiErc4626.sync` sets `_pricePerShare` twice to the same value.
14. `FraxtalERC4626MintRedeemer.updateVaultTknOracle` checks for the `lastVaultTknOracleRead` after the update.

Code partially corrected:

1. Needed for the upgrade.
2. Removed.
3. Left for extra information.
4. Will not fix.
5. Will not fix.
6. The extra call to `sync` was removed.
7. Will not fix. Infrequent use.
8. Was changed to view as it's now reading an immutable value.
9. Removed.
10. Precomputed.
11. Fixed.
12. Fixed.
13. Fixed.
14. Fixed.

7.4 Reentrancy Guard Implementation

Informational **Version 1** **Acknowledged**

CS-FRUP-017

`FraxtalERC4626MintRedeemer` uses `ReentrancyGuard` from OpenZeppelin. Since this is a proxied contract, it is recommended to either use the `ReentrancyGuardUpgradeable` or use the transient version `ReentrancyGuardTransient`.

Acknowledged: Frax won't change this because it would impact the storage layout of the contract.

7.5 Unnecessary Setters in the Constructor

Informational | Version 1 | Acknowledged

CS-FRUP-019

In `SfrxUsd2OracleImplementation`, the constructor executes `_setAllowed` and leaves an active owner on the implementation address. As a good practice, the implementation contract should not have an owner, and the constructor should not execute any setters of state variables.

Acknowledged: Frax will not address the issue.

7.6 Users Can Lose Funds on Rounding

Informational | Version 1 | Risk Accepted

CS-FRUP-020

In `FraxtalERC4626MintRedeemer` users may lose funds if the rounding logic in `withdraw`, `deposit`, `mint`, and `redeem` reduces the computed amount to zero. When this happens, the call burns the user's shares or transfers their assets without delivering any corresponding value.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Changing Beacon Implementation

Note Version 1

The admins of the system should be aware of the behaviors of the beacon factory:

1. During its initialization, the FXBFactory registers the pre-existing FXB contracts. It is important for the admins of the system to note that upon changing the beacon implementation of the rest of the FXBs do not affect the pre-existing FXBs.
2. During the lifetime of the FXBFactory the token used to initialize the FXBS can change with `setToken`. This can lead to divergent behavior between the FXBs, which will still use the same beacon implementation.

8.2 Cross-chain Oracle Desynchronization Enables Arbitrage Opportunities

Note Version 1

The protocol maintains separate price feeds for the native token on Ethereum and its bridged counterpart on Fraxtal. If the two oracles fall out of sync, even briefly, market participants can exploit the resulting price spread:

- Buy the token where it appears cheaper, bridge (or redeem) it, and sell where it appears more expensive.
- Manipulate or front-run the faster-to-update oracle on one chain, then execute trades on the other chain before prices converge.

If the price gap is small relative to the bridge costs and vault fees, the potential profit may be fully negated, effectively voiding the opportunity.

8.3 Function Names After Upgrade

Note Version 1

The upgraded contracts sometimes don't maintain the same interface. This means that contracts that integrate with the protocol and rely on these functions will break. This also applies in the case of publicly accessible state variables, which are renamed with a `DEPRECATED` prefix, that will also modify the respective view functions.

8.4 Future Price and Rounding Errors

Note Version 1

In `LinearRewardsQuasiErc4626` and `SfrxUsd2OracleImplementation`, the internal `_previewPricePerShare` function can be called with a future timestamp. Its output represents a continuously-compounded price-per-share assuming no intermediate `sync()` calls before the target time.

Because each `sync` stores the new price rounded down, an error accumulates with every call. Therefore, the eventual actual on-chain price will be the same or slightly lower, depending on how many times `sync` is executed.

8.5 Price Can Be Stale

Note **Version 1**

Users of `FraxtalERC4626MintRedeemer` must be aware that the view functions are using a stored price, which might not be fresh. This is the case for any `convert*`, `preview*`, and `max*` functions. The last update time can be checked using `last<Vault/Underlying>TknOracleRead`. If the price is stale, the user can update the oracle using one of the `update*Oracle` functions.