Frax Finance Audit Report

(Fraxtal North Star)

Frax Security Cartel

0xleastwood, HickupHH3, Riley Holterhus

March 27, 2025

Contents

1 Introduction **3**

1.1 About Frax Finance . . . 3

1.2 About the Auditors . . . 3

1.3 Disclaimer . . . 3

2 Audit Overview **4**

2.1 Scope of Work . . . 4

2.2 Summary of Findings . . . 4

3 Findings **5**

3.1 Critical Severity Findings . . . 5

3.1.1 Deployment scripts dangerously use `StorageSetter` to allow for contract re-initialization . 5

3.1.2 frxETH approvals can be drained . . . 5

3.2 Low Severity Findings . . . 6

3.2.1 Fee vault contracts do not claim accumulated `frxETH` before hardfork . . . 6

3.2.2 Current `FRAX` deposit & withdrawal methods should be blocked . . . 7

3.2.3 `initTotalSupply` race condition . . . 8

3.3 Informational Findings . . . 9

3.3.1 `ERC20` bridge hardfork edge case . . . 9

3.3.2 Minor codebase improvements . . . 9

3.3.3 `FeeVaultCGT` can use `initiateWithdrawal()` instead of reverting . . . 10

3.3.4 `token` voting power considerations . . . 10

3.4 Gas Optimizations . . . 11

3.4.1 Unused `data` variable . . . 11

1 Introduction

1.1 About Frax Finance

Frax Finance is a DeFi industry leader, featuring several subprotocols that support the FRAX, frxUSD, and frxETH tokens. In early 2024, Frax launched Fraxtal, an optimistic rollup built with the OP stack framework. In 2025, Frax is preparing the North Star hard fork to introduce FRAX as Fraxtal's native gas token. For more details, visit frax.finance.

1.2 About the Auditors

0xleastwood, HickupHH3, and Riley Holterhus are independent smart contract security researchers. All three are Lead Security Researchers at Spearbit, and have a background in competitive audits and live vulnerability disclosures. As a team, they are working together to conduct audits of Frax's codebase, and are operating as the "Frax Security Cartel".

0xleastwood can be reached on Twitter at @0xleastwood, HickupHH3 can be reached on Twitter at @HickupH, and Riley Holterhus can be reached on Twitter at @rileyholterhus.

1.3 Disclaimer

This report is intended to detail the identified vulnerabilities of the reviewed smart contracts and should not be construed as an endorsement or recommendation of any project, individual or entity. While the authors have made reasonable efforts to detect potential issues, the absence of any undetected vulnerabilities or issues cannot be guaranteed. Additionally, the security of the smart contracts may be affected by future changes or updates. By using the information in this report, you acknowledge that you are doing so at your own risk and that you should exercise your own judgment when implementing any recommendations or making decisions based on the findings. This report has been provided on an "as-is" basis and DOES NOT CONSTITUTE A GUARANTEE OR WARRANTY OF ANY FORM.

2 Audit Overview

2.1 Scope of Work

From February 10, 2025 through February 21, 2025, the Frax Security Cartel audited the Fraxtal North Star hard fork smart contracts. The scope of this review was in Frax's dev-fraxchain-contracts GitHub repository on commit hash ff6684ae129e3925e8c210e96750a70ffb29c1ac, specifically in the `src/contracts/Fraxtal/` directory.

The audit encompassed the smart contracts associated with the North Star hard fork, which migrates to FRAX (formerly FXS) as Fraxtal's native gas token. The main Fraxtal portal contract and several periphery contracts were reviewed during the audit:

- src/contracts/Fraxtal/L1/OptimismPortalCGT.sol
- src/contracts/Fraxtal/L1/SystemConfigCGT.sol
- src/contracts/Fraxtal/L1/L1StandardBridgeCGT.sol
- src/contracts/Fraxtal/L1/L1CrossDomainMessengerCGT.sol
- src/contracts/Fraxtal/L2/L1BlockCGT.sol
- src/contracts/Fraxtal/L2/L2CrossDomainMessengerCGT.sol
- src/contracts/Fraxtal/L2/L2StandardBridgeCGT.sol
- src/contracts/Fraxtal/L2/L2ToL1MessagePasserCGT.sol
- src/contracts/Fraxtal/L2/BaseFeeVaultCGT.sol
- src/contracts/Fraxtal/L2/FeeVaultCGT.sol
- src/contracts/Fraxtal/L2/L1FeeVaultCGT.sol
- src/contracts/Fraxtal/L2/SequencerFeeVaultCGT.sol

Additionally, the hard fork includes a migration of Fraxtal's L2 tokens. The FXS token will transition from `OptimismMintableERC20` to `ERC20ExPPOMWrapped` and will be renamed to wFRAX. The wfrxETH token will migrate from a WETH-like contract to `ERC20ExWrappedPPOM` and will be renamed to frxETH. The following files were in scope for this part of the review:

- src/contracts/Fraxtal/universal/ERC20ExPPOMWrapped.sol
- src/contracts/Fraxtal/universal/ERC20ExWrappedPPOM.sol

2.2 Summary of Findings

Each finding from the audit has been assigned a severity level of "Critical", "High", "Medium", "Low" or "Informational". These severities aim to capture the impact and likelihood of each potential issue. Gas optimization findings have been included in their own section of the report.

In total, **10 findings** were identified. This includes **2 critical**, **3 low**, and **4 informational** severity findings, as well as **1 gas optimization** finding. All issues have either been directly addressed by the Frax team, or have been acknowledged as acceptable behavior.

3 Findings

3.1 Critical Severity Findings

3.1.1 Deployment scripts dangerously use `StorageSetter` to allow for contract re-initialization

Description: Currently, the hardfork upgrade setup looks like the following:

- Foundry scripts generate Gnosis Safe batch data.
- This is split up into json files containing relevant transactions for each stage of the hardfork.
- However, `SHOULD_EXECUTE` is set to `true` which indicates Gnosis Safe transactions will be executed from within the script instead of utilising the json data.

There is a serious concern here in the the potential lack of atomicity that could allow for a malicious actor to step in during the clearing of the relevant initialization storage slots to take admin ownership of the proxy contracts. This is because the `StorageSetter` contract allows for the arbitrary writing of slots. Effectively, by taking ownership of the proxy contract, the implementation can be upgraded to a malicious contract and used to drain the bridge.

Recommendations: Consider being explicit about what slots can be written by the `StorageSetter` contract and ensure the Gnosis Safe transaction batch is executed atomically.

Frax: Fixed in commit 29eb5c4.

Frax Security Cartel: Verified. The `StorageSetter` contract has been renamed to `StorageSetterRestricted` and is now only useful for clearing the zero slot containing related initialization variables. However, the use of this contract is mostly protected because sensitive upgrade steps are batched and should be executed atomically through the Frax team's Gnosis Safe multisig.

3.1.2 frxETH approvals can be drained

Description: The Fraxtal North Star hardfork will update the Fraxtal portal proxy to use a new portal implementation. One key difference between the implementations is that the new one uses FRAX (formerly known as FXS) as the native/gas token, instead of frxETH which is currently used.

In both implementations, users approve the portal to use their tokens, and the portal calls `transferFrom()` when processing a deposit.

Since the portal also handles L2 to L1 messages, it has the ability to make arbitrary calls with arbitrary calldata. To mitigate potential exploits, the old implementation explicitly prevented frxETH from being the target of a call, and the new implementation has the same restriction on the `token` (FRAX in this case):

```
// Old implementation:
function finalizeWithdrawalTransaction(/* ... */) /* ... */ {
    // ...
    require(_tx.target != FRXETH, "FraxchainPortal: can not target frxETH");
    // ...
}
```

```
// New implementation:
function finalizeWithdrawalTransaction(/* ... */) /* ... */ {
    // ...
    if (token == Constants.ETHER) {
        // ...
    } else {
        // ...
        if (_tx.target == token) revert BadTarget();
        // ...
    }
    // ...
}
```

However, note that the new implementation has removed the check that prevents frxETH from being the target of a call. This introduces an exploit.

After the hardfork, users will still have lingering frxETH approvals to the portal. If an attacker initiates an L2 to L1 withdrawal that calls `frxETH.transferFrom(victim, attacker, victim balance)`, the new portal implementation will not revert. If such a withdrawal was actually executed, it would transfer frxETH from the victim's wallet to the attacker.

Recommendation: Add a check in the new implementation to explicitly prevent frxETH from being the target of a withdrawal, as was done in the old implementation.

Note that an alternative mitigation is to have all users manually revoke their frxETH allowances to the portal proxy. However this would be impractical given the number of users. Tools like reverse-revoke.netlify.app can help visualize all of the users that have given approval to a given smart contract.

Frax: Fixed in commit f2d7059 and commit adc27d1.

Frax Security Cartel: Verified. Note that this check introduces a new `FRXETH` storage variable, and this storage variable has been placed after a new `uint256[50] __gap` to accommodate other storage variables added to the Optimism codebase in the future.

3.2 Low Severity Findings

3.2.1 Fee vault contracts do not claim accumulated `frxETH` before hardfork

Description: Any contract which inherits `FeeVaultCGT` will have the zero slot overwritten with `_initialized` and `_initializing` which are packed into the same slot. Prior to the hardfork, contracts inheriting `FeeVaultCGT` are not initializable and therefore the contract only has a single slot to track `totalProcessed`. However, these same contracts will be upgraded to effectively overwrite `totalProcessed` with `_initialized` and `_initializing` and add other state variables. Once upgraded, it is no longer possible to claim any `frxETH` which has accumulated in the relevant contracts (~48.5 `ETH`).

Affected contracts include:

- BaseFeeVaultCGT
- L1FeeVaultCGT
- SequencerFeeVaultCGT

Recommendations: Some decision needs to be made on whether to increment the totalProcessed amount if transferring out the frxETH balance held in the contract. If incremented, it seems that totalProcessed would no longer be an accurate representation as it is denominated in amounts from two different native tokens.

Frax: Fixed in commit 6cd3472 where frxETH is withdrawn on initialize() and totalProcessed zeroed.

Frax Security Cartel: Verified. The fee vaults will send out frxETH balance on re-initialization.

3.2.2 Current FRAX deposit & withdrawal methods should be blocked

Description: The L1StandardBridge still allows bridging FRAX tokens via bridgeERC20() and bridgeERC20To() methods, even though these should be blocked in favor of using OptimismPortalCGT.depositERC20Transaction(). While the transaction will ultimately fail when trying to mint on L2 (since the L2 token is upgraded to ERC20ExPPOMWrapped which has REMOTE_TOKEN deprecated), this creates a confusing user or developer experience.

POC: The following tests demonstrate the mentioned issue: 1. Users can still call depositERC20() on L1StandardBridge with FRAX 2. The transaction will revert on L2 due to arithmetic underflow when trying to mint

```
function test_depositERC20WorksForCGT() public {
  // deal user with 1000 FRAX
  deal(l1FRAX, user, 1000 ether);
  vm.startPrank(user);
  IERC20(l1FRAX).approve(address(l1StandardBridge), 1000 ether);
  // assert custom gas token is l1FRAX
  (address l1GasToken, ) = ISystemConfig(l1StandardBridge.systemConfig()).gasPayingToken();
  assertEq(l1GasToken, l1FRAX, "L1 gas token should be l1FRAX");
  l1StandardBridge.depositERC20(
    l1FRAX,
    address(l2FRAX),
    1e18,
    0,
    ""
  );
}

function test_mintShouldFailForWFRAX() public {
  // message from test_depositERC20WorksForCGT
  bytes memory message = bytes.concat(
      hex"0166a07a",
      hex"000000000000000000000000fc00000000000000000000000000000000000002",
      hex"0000000000000000000000008301eb65ded23422ea8eeb64bf33d40553d32c1f",
      hex"00000000000000000000000026ff577818a40d1333944b7492e23619303f5d84",
      hex"00000000000000000000000026ff577818a40d1333944b7492e23619303f5d84",
      hex"0000000000000000000000000000000000000000000000000de0b6b3a7640000",
      hex"00000000000000000000000000000000000000000000000000000000000000c0",
      hex"0000000000000000000000000000000000000000000000000000000000000000"
  );
  uint256 nonce = 1766847064778384329583297500742918515827483896875618958121606201292619781;
  address sender = 0xE6E340D132b5f46d1e472DebcD681B2aBc16e57E;
  address target = 0x4200000000000000000000000000000000000010;
  address aliasedOtherMessenger = AddressAliasHelper.applyL1ToL2Alias(address(
      l2CrossDomainMessenger.otherMessenger()));
  vm.startPrank(aliasedOtherMessenger);
  // result: [Revert] panic: arithmetic underflow or overflow (0x11)
  // emit FailedRelayedMessage(msgHash: 0
      xe04b515ff46e2ea14ce3a5b7d92b7faaf83385262df37645e7086a0cca074f20)
  l2CrossDomainMessenger.relayMessage(
    nonce,
    sender,
    target,
    0, // value
    0, // minGasLimit
    message // message
  );
}
```

Recommendation: Modify `L1StandardBridge` to block bridging of `FRAX` tokens via `bridgeERC20()` and `bridgeERC20To()`.

Frax: Fixed in commit 60e2352, commit adc27d1 and commit ad4f0fc.

Frax Security Cartel: Verified.

3.2.3 `initTotalSupply` race condition

Description: As part of the North Star hardfork, the `wfrxETH` L2 contract (0xfc00000000000000000000000000000000000006) will be placed behind a proxy, upgraded to the `ERC20ExWrappedPPOM` implementation, and renamed to `frxETH`. Since the current `wfrxETH` contract does not have a `_totalSupply` variable like `ERC20ExWrappedPPOM` does, the `initialize()` function in the upgrade will explicitly set `_totalSupply`:

```
  function initialize(/* ... */, uint256 _initTotalSupply, /* ... */) /* ... */ {
          // ...

          // Overwrite _totalSupply storage
          //-------------------------------------
          assembly {
              sstore(9, _initTotalSupply)
          }

          // ...
  }
```

It's worth noting that if the upgrade is executed via a Gnosis Safe transaction, there may be a delay between when `_initTotalSupply` is hardcoded in the Gnosis Safe calldata, and when the transaction is actually executed. If such a gap existed, then any users depositing or withdrawing `wfrxETH` in the meantime would cause the final `_initTotalSupply` to be slightly wrong.

This could be a problem if `_initTotalSupply` is too low. In this case, not all `frxETH` can be withdrawn back to L1, since the `_totalSupply` would eventually underflow.

Recommendation: Consider adding a function to `ERC20ExWrappedPPOM` that allows `_totalSupply` to be adjusted after initialization. This function could increment or decrement `_totalSupply` by a constant value to correct any mistakes due to race conditions.

Frax: Fixed in commit ece2ce1 and commit 9a52a71.

Frax Security Cartel: Verified.

3.3 Informational Findings

3.3.1 `ERC20` bridge hardfork edge case

Description: The Fraxtal North Star hardfork will upgrade wfrxETH on L2 (0xFc0000000000000000000000000000000000006), renaming it to frxETH and transitioning it into an Optimism mintable remote token for L1 frxETH (0x5E8422345238F34275888049021821E8E08CAa1f). This change will also impact how the ERC20 bridge handles deposits and withdrawals on both ends.

An interesting edge case to consider is what happens if an L2 to L1 bridge of wfrxETH to frxETH is initiated *before* the hardfork. The transaction would succeed on L2, as the bridge contract would simply `transferFrom()` the tokens and update its `deposits` mapping. When the corresponding message is executed on L1, it would fail at first, since the L1 `deposits` mapping would be zero. However, after the hardfork, assuming the upgrade will treat all frxETH on L2 as having been bridged through the ERC20 bridge, the L1 `deposits` mapping would be updated, allowing the withdrawal to succeed.

This behavior doesn't seem to be exploitable. The main unintended effect would be that some L2 frxETH remains in the L2 bridge contract. This is unexpected since after the hardfork, L2 frxETH will be handled via minting and burning, so no L2 frxETH will exist in the bridge otherwise.

Recommendation: This finding is provided for informational purposes, and no code changes seem necessary. Monitoring L2 to L1 bridge transactions before the upgrade is recommended to detect any unexpected behavior.

Frax: Acknowledged. Also, we will tell people to clear out and finalize bridging transactions.

Frax Security Cartel: Acknowledged.

3.3.2 Minor codebase improvements

Description: There are some minor natspec and spelling errors, see the recommendations for suggested fixes.

Recommendations:

```
- receipient
+ recipient

- nonexistant
+ non-existent

- /// @param timelock_address Address of the removed timelock
+ /// @param timelock_address Address of the new timelock
```

Frax: Fixed in commit 60e2352, commit 9a52a71 and commit 001d202.

Frax Security Cartel: Verified.

3.3.3 `FeeVaultCGT` can use `initiateWithdrawal()` instead of reverting

Description: The `FeeVaultCGT` currently reverts withdrawals to L1 due to `L2ToL1MessagePasser` requiring `msg.value` to be zero. However, Optimism's version uses `initiateWithdrawal()`, which could be adopted here to enable L1 withdrawals.

Recommendation: Update the `FeeVaultCGT` to use `L2ToL1MessagePasser.initiateWithdrawal()` for L1 withdrawals:

```
} else {
-  // Because of the custom gas token, you cannot withdraw to L1 because L2ToL1MessagePasser must
    have zero msg.value
-  revert CannotWithdrawToL1();
+ IL2ToL1MessagePasser(payable(Predeploys.L2_TO_L1_MESSAGE_PASSER)).initiateWithdrawal{ value:
    value }({
+    _target: RECIPIENT,
+    _gasLimit: WITHDRAWAL_MIN_GAS,
+    _data: hex""
+ });
}
```

Frax: Fixed in commit 60e2352.

Frax Security Cartel: Verified.

3.3.4 `token` voting power considerations

Description: After the North Star hardfork, the Fraxtal portal contract will hold all `token` bridged to L2. The `token` after the hardfork will be `FRAX` (formerly known as `FXS`).

Note that the FXS contract includes voting logic based on balance, with functions like `getCurrentVotes()` and `getPriorVotes()`. Since the portal contract can make arbitrary calls to most addresses unless explicitly restricted, it would be problematic if there are any contracts using `FXS` voting. If such a contract existed, it should be blocked as a withdrawal target, since an attacker could send an L2 to L1 message to "vote" using the portal's balance.

A Dune query of all contracts that have previously called `getCurrentVotes()` or `getPriorVotes()` on the `FXS` contract found only one match: the `GovernorAlpha` contract at address 0xd74034c6109a23b6c7657144cacbbbb82bdcb00e. However, this appears to be a deprecated contract from a previous iteration of the governance system.

Recommendation: After discussing with the Frax team, it was determined that `FXS` voting is unused and will potentially be disabled in the future. As a result, no code changes are required, and this finding is provided for informational purposes only.

Also, based on the analysis showing that only one deprecated contract has ever called the `FXS` voting functions, disabling vote tracking on `FXS` via `toggleVotes()` could be considered to reduce gas costs.

Frax: The `toggleVotes()` function on `FXS` will be called.

Frax Security Cartel: Verified.

3.4 Gas Optimizations

3.4.1 Unused `data` variable

Description: In `L1BlockCGT`, there's an unused variable `data` in the assembly block which is loaded from `calldata` but never utilized.

Recommendation: Remove the `data` variable.

Frax: Fixed in commit d14da1f.

Frax Security Cartel: Verified.