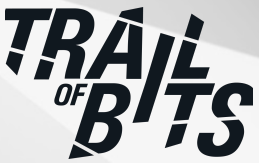

Frax Finance: Frax Bonds, Savings Frax, Frax Ether Redemption Queue, Frax Oracles

Security Assessment

October 27, 2023

Prepared for:
Sam Kazemian
Frax Finance

Prepared by: **Robert Schneider and Richie Humphrey**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Anne Marie Barry, Project Manager
annemarie.barry@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Robert Schneider, Consultant
robert.schneider@trailofbits.com

Richie Humphrey, Consultant
richie.humphrey@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 7, 2023	Pre-project kickoff call
September 15, 2023	Status update meeting #1
September 25, 2023	Delivery of report draft
September 25, 2023	Report readout meeting
October 27, 2023	Delivery of comprehensive report with fix review appendix

Executive Summary

Engagement Overview

Frax Finance engaged Trail of Bits to review the security of its Frax Bonds, Savings Frax, Frax Ether Redemption Queue, and Frax Oracles projects.

A team of two consultants conducted the review from September 11 to September 22, 2023, for a total of four engineer-weeks of effort. Our testing efforts focused on ensuring that the `frax-oracles` and `frax-eth-redemption-queue` codebases are configured properly on deployment, and that the `frax-bonds`, `savings-frax-erc4626`, and other codebases related to the projects under audit do not contain exploitable bugs. With full access to the source code and documentation, we performed static and dynamic testing of the codebases listed above, using automated and manual processes. We did not review any oracles outside of the audit scope, nor did we review the factory contracts in `frax-bonds` that were not included in the audit scope.

Observations and Impact

During this audit, we reviewed four separate projects that are isolated from one another and have distinct areas of concern. Two of the four projects, `frax-ether-redemption-queue` and `frax-oracles`, had already been deployed. For those two projects, we reviewed the deployment configuration and found an instance in which an oracle had been deployed with the wrong base token address (TOB-FBSRO-2). In the other two projects, `frax-bonds` and `savings-frax-erc4626`, most of the features and functionality are well implemented, commented, and clear. However, we did find two glaring oversights in the `SlippageAuction` contract (TOB-FBSRO-3, TOB-FBSRO-4) that we feel could have been easily caught if a more exhaustive and thorough unit test suite had been developed for that contract.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Frax Finance take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Enhance the unit test suite.** Remove commented out sections from the unit test suite. Enhance the current tests to include in-depth testing of all critical paths and functions to ensure they behave as expected. Give careful consideration to edge cases and user-provided inputs to cover expected behavior when given unexpected inputs.

- **Further develop system invariants that are expected to hold true against the system state.** These invariants should specify the expected system behavior and the global system state when certain functions are called.
- **Add fuzz testing.** Incorporating system invariants into fuzz tests can help identify additional edge cases that the system may not be able to handle. This approach can help uncover hidden bugs that may not be immediately apparent.
- **Develop a post-deployment checklist.** Prior to deploying new contracts, draw up a checklist of the expected post-deployment system configuration. Once a new contract system is deployed, go over each item on the checklist to ensure that the deployed state matches the expected state. That way, the Frax Finance development team can catch misconfigured contracts early.

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Medium	1
Low	1
Informational	1

CATEGORY BREAKDOWN

Category	*Count*
Configuration	3
Data Validation	1
Timing	1

Project Goals

The engagement was scoped to provide a security assessment of the Frax Bonds, Savings Frax, Frax Ether Redemption Queue, and Frax Oracles projects. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are appropriate access controls in place for the various roles in the system?
- Is there a risk of race conditions, such as front-running? If so, what mitigations are in place to prevent them?
- Are there any denial-of-service attack vectors?
- Can a user's funds become trapped in the system?
- Is the system vulnerable to reentrancy attacks?
- Does the system implementation match the specification described by EIP-4626, EIP-20, and EIP-721?
- Were the contracts that have already been deployed configured properly?
- Are all inputs properly validated before being used? Do the contracts correctly handle both expected and unexpected inputs?
- Do the contracts include descriptive error messages and `require` statements that facilitate debugging and understanding of the contracts' behavior?

Project Targets

The engagement involved a review and testing of the targets listed below.

frax-bonds

Repository	https://github.com/FraxFinance/frax-bonds
Version	`9c4be497c330f56f635fd699390f3f979ec37f0d`
Type	Solidity
Platform	Ethereum

savings-frax-erc4626

Repository	https://github.com/FraxFinance/savings-frax-erc4626
Version	`0761cf8f88da298c600609540a89dbfb11ed7c59`
Type	Solidity
Platform	Ethereum

frax-ether-redemption-queue

Repository	https://github.com/FraxFinance/frax-ether-redemption-queue
Version	`868bff484905d5bdfc011d95f2d417d247ee9d57`
Type	Solidity
Platform	Ethereum

frax-oracles

Repository	https://github.com/FraxFinance/frax-oracles
Version	`bd56532a3c33da95faed904a5810313deab5f13c`
Type	Solidity
Platform	Ethereum

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

Frax Bonds

- **`FXB.sol`:** This contract is akin to a zero coupon bond in that it guarantees the holder 1 FRAX at the maturity date. We attempted to find ways that an attacker could drain the pool, receive FRAX early, or bypass the access control on the `mint()` function. Additionally, the `burn()` function's dependency on a specified maturity date necessitated a thorough examination to ensure that the maturity date logic was implemented securely and correctly to prevent premature or delayed token burns.

- **`SlippageAuction.sol`:** This contract auctions off 1 ERC-20 "sell" token for a "buy" token at a price that decreases over time. We reviewed the owner privileges to start or stop auctions to determine whether this power could be misused or transferred maliciously. We also thoroughly examined the contract's pricing mechanism, which decreases over time until a specified `minPrice` and adjusts prices when purchases cause price elevation, to ensure it operates as intended and does not contain exploitable bugs. The implementation of the `swap` function, resembling the functionality of the `UniswapV2Pair` contract, was scrutinized particularly regarding its flash loan capabilities and its integration into routers to ensure that it does not introduce vulnerabilities that could enable reentrancy attacks or other flash loan attack vectors. Additionally, the logic that handles slippage and its effect on buyers was reviewed to confirm that it is implemented securely and functions as intended without negatively impacting users or the contract's overall functionality.

Savings Frax

- **`LinearRewardsERC4626.sol`:** The contract was analyzed to verify the accuracy and consistency of its mechanism for linear reward distribution over a specified reward cycle. The examination focused on ensuring that the distribution rate remains constant per block whenever a transaction occurs. Concerns were raised about the need to queue rewards for distribution, either by calling a sync function or upon the first interaction after the conclusion of a previous cycle. The logic governing the synchronization process and the initiation of new reward cycles was carefully reviewed to ensure that it is securely implemented, free from any potential loopholes that could be exploited to manipulate reward distribution.

- **`SavingsFrax.sol`:** The contract inherits from `LinearRewardsERC4626.sol` and introduces a maximum reward rate. The review focused on overriding the `calculateRewardsDistribution` function. It is crucial to ensure that the

overriding function effectively caps the rewards distribution rate as intended, without introducing unexpected behaviors or vulnerabilities. Another significant concern is the ability of a timelock address to update the maximum distribution rate. This required a thorough review to confirm that only authorized addresses can execute this function and that appropriate timelock delays are enforced to prevent sudden or malicious changes to the reward distribution rate. The implementation of this mechanism was inspected to ensure it follows security best practices and provides a robust and secure framework for managing the rate of reward distribution.

Frax Ether Redemption Queue

- **`FraxEtherRedemptionQueue.sol`:** The contract outlines a redemption queue for exchanging frxETH for ether. The trust placed on admin accounts raised concerns, requiring a thorough review of the access controls and admin privileges. The issuance of NFTs representing redemption information was reviewed to ensure that they are issued securely and accurately. The logic behind the issuance and data encapsulation was examined for consistency and accuracy. The contract introduces a configurable redemption waiting period and fee. Mechanisms for configuring these settings were examined for opportunities to exploit them. The option for users to exit the queue early and reclaim frxETH at an additional cost was scrutinized to ensure it is implemented securely. The fee structures, including redemption fees and early exit fees, were reviewed for fairness and transparency. The audit aimed to ensure a balanced and secure operation of the `FraxEtherRedemptionQueue` contract, protecting user interests and adhering to stated functionalities.

Frax Oracles

- Several oracles were among the contracts that had already been deployed. We gave special attention to the configuration that these contracts were deployed with. We looked holistically at the incorporation of three services—Curve, Uniswap, and Chainlink—to provide accurate and robust pricing information. We considered oracle manipulation and sandwich attacks and how they might impact the system. We reviewed safety measures put in place to ensure that bad pricing data is not used. We considered peculiarities of the Arbitrum network, as opposed to the Ethereum network, to ensure that they pose no implications for these oracle contracts.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- In the `frax-oracles` repository, only the following contracts were in scope for this audit: `FrxEthEthDualOracle`, `FrxEthFraxOracle`, `SfrxEthEthDualOracle`, `SfrxEthFraxOracle`, `ArbitrumBlockHashRelay`, `ArbitrumBlockHashProvider`, `FrxEthFraxOracle`, `MerkleProofPriceSource`, and `StateRootOracle`.
- The `StateProofVerifier` and `MerklePatriciaProofVerifier` contracts were audited (not by Trail of Bits) and were not considered in scope for this audit.
- In the `frax-bonds` repository, `FXBFactory` and `SlippageAuctionFactory` contracts were not considered in scope for this audit.
- Assessment of any incident response plan was considered out of scope for this audit.
- A deep review and reconciliation of the inheritance structure of the codebases was not performed.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The codebases adhere to safe math practices by using Solidity version 0.8 and above, which includes safe math operations. The absence of unnecessary unchecked arithmetic minimizes overflow and underflow risks. Thorough documentation improves code comprehensibility in complex mathematical areas. However, there is a lack of in-depth testing for math-specific components, which may result in undetected errors. Implementing extensive math-specific testing and fuzz testing would enhance the security of the smart contract system.	**Moderate**
Auditing	The codebases effectively use events for critical operations and state changes, enabling system monitoring through third-party integrations. However, there is a lack of documentation explaining the purpose, utilization, and assumptions of events. This can hinder auditing and logging operations for developers and external auditors who are unfamiliar with the system. The absence of monitoring documentation for reviewing logs in the event of a failure is also a notable oversight. Providing comprehensive documentation on events and the monitoring and auditing process is crucial for a transparent and auditable codebase. This would facilitate troubleshooting and identification of operational anomalies, if any.	**Moderate**
Authentication / Access Controls	The codebases show a careful implementation of access control mechanisms for all privileged functions, following the principle of least privilege. This practice ensures that each component, including users and systems, has the necessary access permissions for its tasks, reducing the attack surface and improving the security of the smart	**Satisfactory**

	contract system. However, although there are tests covering privileged function access controls, there is room to expand and enhance these tests to create a stronger authentication and access control framework.	
Complexity Management	The bridging of oracle prices from layer one to layer two networks is inherently complex, especially with the reliance on Merkle proofs for verification. The use of three different oracle systems (Curve, Uniswap, Chainlink) adds more complexity and could compromise system reliability if one oracle goes down. The inherited code in the oracle contracts and `SavingsFrax` contract also adds complexity and makes code comprehension and maintenance difficult. Streamlining the architecture or providing extensive documentation and a comprehensive test suite can help manage these complexities. Addressing these concerns will improve codebase navigation, maintainability, and system reliability in the smart contracts.	**Moderate**
Decentralization	Upgradeable contracts are not used in the system, which helps to decentralize the system and reduce centralized control. Governance contracts are used to control the system's parameters, aligning well with the decentralized nature of blockchain systems. The implementation of a timelock controller enables users to opt out of updates. Additionally, the codebases employ a secure two-step process for ownership transfer, minimizing the likelihood of mistakes during this critical procedure. It is important to thoroughly document these mechanisms to foster transparency and build trust within Frax Finance's decentralized ecosystem.	**Satisfactory**
Documentation	The codebases follow well-established documentation standards with NatSpec comments, aiding code comprehension and maintenance. Inline code comments clarify complex sections. Audit-specific documentation that helped facilitate an efficient audit was provided. However, some NatSpec comments are outdated, which could mislead developers and auditors. It is crucial to maintain and update documentation alongside code changes. Reviewing and updating documentation would ensure accuracy and understanding of the system.	**Satisfactory**

Transaction Ordering Risks	The codebases minimize risks associated with user transaction ordering, ensuring fairness and integrity. Tamper-resistant oracles further protect against adversarial manipulations and data reliance. This approach mitigates transaction ordering risks and safeguards the system against adversarial exploitation. Thorough analysis and documentation of the results could further strengthen user confidence against risks.	**Satisfactory**
Low-Level Manipulation	The code does not use low-level manipulation. The contracts that do use it are dependencies, which were not reviewed for correctness.	**Not Applicable**
Testing and Verification	Several issues identified during this audit could have been prevented by a more detailed and robust unit test suite. The current test suite mainly focuses on testing the system's happy paths, while tests for more comprehensive coverage were either commented out or not fully developed. It would be beneficial to develop a fuzz test suite for critical components, and running the entire test suite in the CI pipeline could help improve the bug identification and triaging process.	**Weak**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	FRAX tokens mistakenly sent to FXB cannot be retrieved	Configuration	**Informational**
2	Oracle deployed with misconfigured base token	Configuration	**Low**
3	Array assignment to index out of bounds breaks swap functions	Configuration	**High**
4	Incorrect order of arguments passed to swap function triggers revert	Data Validation	**High**
5	collectRedemptionFees can be called before feeRecipient has been set	Timing	**Medium**

Detailed Findings

1. FRAX tokens mistakenly sent to FXB cannot be retrieved	
Severity: **Informational**	Difficulty: **Medium**
Type: Configuration	Finding ID: TOB-FBSRO-1
Target: `frax-bonds/src/contracts/FXB.sol`	

Description
The FXB contract does not have a way to recover tokens sent directly to it. Users who mistakenly send FRAX tokens to the contract will lose their funds.

During the minting process, FRAX tokens are transferred from the caller's account to the FXB contract. At the same time, an equal amount of FXB tokens are minted and deposited into the caller's account.

```
function mint(address _to, uint256 _amount) public {
        // NOTE: Allow minting after expiry

        // Effects: Give the FXB to the recipient
        _mint({ account: _to, amount: _amount });

        // Interactions: Take 1-to-1 FRAX from the user
        FRAX.transferFrom({ sender: msg.sender, recipient: address(this), amount:
_amount });
}
```

Figure 1.1: The `mint` method of the FXB contract (FXB.sol#L101-L109)

This feature of the FXB contract's minting process increases the risk that users will mistakenly send FRAX tokens directly to the FXB contract without going through the `mint` function; as a result, no FXB tokens would be minted for that user. Since there is no mechanism for retrieving tokens from the FXB contract, the affected user would lose those tokens. Additionally, there is a reputational risk for Frax Finance because it controls both tokens, and the frozen FRAX would effectively be burned.

Exploit Scenario
Bob holds FRAX tokens and wants to mint FXB. He prepares a call to the `approve` method on the FRAX token contract before he calls the `mint` function of the FXB contract. Bob accidentally calls the `transfer` method instead of the `approve` method. Bob's FRAX tokens are now stuck in the FXB contract.

Recommendations

Short term, implement a retrieval function that uses the `totalSupply` value of FXB to determine the amount of excess FRAX tokens in the contract. This retrieval function should enable the recovery of the difference through Frax Finance's governance process.

Long term, when making design decisions that deviate from standard usage, consider edge cases and unlikely user inputs. Building a robust suite of unit tests that considers uncommon edge cases can help refine cases that the system should prevent or accommodate, which may not be immediately clear. Clearly warn end users in the documentation that they must not send FRAX tokens directly to the FXB contract.

2. Oracle deployed with misconfigured base token	
Severity: **Low**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-FBSRO-2
Target: `SfrxEthEthDualOracle.sol`	

Description
The `SfrxEthEthDualOracle` contract was deployed to the mainnet with the base token values (both `BASE_TOKEN_0` and `BASE_TOKEN_1`) set to `0x5E8422…8CAa1f`, the address of frxETH, instead of `0xac3E01…bbe38F`, the address of sfrxETH.

Exploit Scenario
A problem is discovered with the `sfrxETH` contract that does not affect frxETH. The price of sfrxETH drops, but because the oracle is set incorrectly, Eve is able to take a loan against sfrxETH collateral.

Recommendations
Short term, redeploy the oracle with the correct configuration.

Long term, track the configuration settings of deployed contracts in a shared off-chain registry. Use the registry in the deployment scripts and for post-deployment tests.

3. Array assignment to index out of bounds breaks swap functions

Severity: **High**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-FBSRO-3
Target: `frax-bonds/src/contracts/SlippageAuction.sol`	

Description

The last two lines of both swap functions, `swapTokensForExactTokens` and `swapExactTokensForTokens`, incorrectly attempt to write to out-of-bounds array indices, which causes the functions to revert.

```
function swapTokensForExactTokens(uint256 _amountOut, uint256 _amountInMax,
address[] calldata _ignored,
    address _to,
    uint256 _deadline
) external returns (uint256[] memory _amounts) {

    if (block.timestamp > _deadline) revert Expired();

    (uint256 _amountIn, , ) = getAmountIn(_amountOut);

    if (_amountIn > _amountInMax) revert ExcessiveInputAmount({ minIn: _amountInMax,
actualIn: _amountIn });
    IERC20(BUY_TOKEN).safeTransferFrom({ from: msg.sender, to: address(this), value:
_amountIn });

    this.swap(0, _amountOut, _to, new bytes(0));
    _amounts[0] = _amountIn;
    _amounts[1] = _amountOut;
}
```

Figure 3.1: contracts/SlippageAuction.sol#408-425

Both functions use a named return variable `_amounts`, which is initialized as a zero length array by default. During the functions' execution, the `_amounts` variable is not referenced and the memory slot it resides in is not altered in any way. The intent of the last two lines is to add the incoming and outgoing amounts as the values for the first two elements of the `_amounts` array. However, because it is still a zero length array, those lines always access the array out of bounds, causing a runtime error.

Exploit Scenario

Alice wants to buy some Frax bonds, but when she calls the swap function, it reverts.

Recommendations

Short term, prior to the last two lines, have the functions initialize a static array with the new keyword and the expected length (e.g., `_amounts = new uint256[](2);`).

Long term, ensure that the tests cover all core functions of the protocol.

4. Incorrect order of arguments passed to swap function triggers revert

Severity: **High**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FBSRO-4
Target: `frax-bonds/src/contracts/SlippageAuction.sol`	

Description

One of the core swap functions, `swapExactTokensForTokens`, will always revert because it passes arguments to the `swap` function in the incorrect order.

```
function swapExactTokensForTokens(
    uint256 _amountIn,
    uint256 _amountOutMin,
    address[] calldata _ignored,
    address _to,
    uint256 _deadline
) external returns (uint256[] memory _amounts) {
    if (block.timestamp > _deadline) revert Expired();
    (uint256 _amountOut, , ) = getAmountOut(_amountIn, true);
    if (_amountOut < _amountOutMin) {
        revert InsufficientOutputAmount({ minOut: _amountOutMin, actualOut:
_amountOut });
    }
    IERC20(BUY_TOKEN).safeTransferFrom({ from: msg.sender, to: address(this), value:
_amountIn });
    this.swap(_amountOut, 0, _to, new bytes(0));
```

Figure 4.1: contracts/SlippageAuction.sol#L434-L446

The last line of the function calls `swap` on the current contract and passes `amountOut` as the first argument and `0` as the second. The `swap` function's first parameter is the amount of buy tokens that should go out. This amount should always be zero because this auction does not transfer out any buy tokens. The second parameter is the amount of sell tokens going out. These arguments are reversed.

```
function swap(uint256 _buyTokenOut, uint256 _sellTokenOut, address _to, bytes
calldata _data)
```

Figure 2.2: The function signature for swap( ) (contracts/SlippageAuction.sol#L343)

As a result, the swap function fails on the first line of the function during the check ensuring that `_sellTokenOut` is not zero.

```
FAIL. Reason: InsufficientOutputAmount({ minOut: 1, actualOut: 0 })
```

Figure 4.3: The runtime error that occurs when swap is called

Recommendations

Short term, fix the order of the arguments passed to swap in the `swapExactTokensForTokens` function. Consider using named arguments to reduce problems of this nature.

Long term, ensure that the tests cover all core functions of the protocol.

5. collectRedemptionFees can be called before feeRecipient has been set

Severity: **Medium**	Difficulty: **High**
Type: Timing	Finding ID: TOB-FBSRO-5
Target: `frax-ether-redemption-queue/src/contracts/FraxEtherRedemptionQueue.sol`	

Description

The `collectRedemptionFees` function does not check to ensure that the `feeRecipient` address has been set. As a result, fees can mistakenly be sent to the default zero address.

```
   function collectRedemptionFees(uint128 _collectAmount) external {
        // Make sure the sender is either the timelock or the operator
        _requireIsTimelockOrOperator();

        uint128 _unclaimedFees = redemptionQueueAccounting.unclaimedFees;

        // Make sure you are not taking too much
        if (_collectAmount > _unclaimedFees) revert
ExceedsCollectedFees(_collectAmount, _unclaimedFees);

        // Decrement the unclaimed fee amount
        redemptionQueueAccounting.unclaimedFees -= _collectAmount;

        // Interactions: Transfer frxEth fees to the recipient
        IERC20(address(FRX_ETH)).safeTransfer({ to: feeRecipient, value:
_collectAmount });

        emit CollectRedemptionFees({ recipient: feeRecipient, collectAmount:
_collectAmount });
    }
```

Figure 5.1: The `collectRedemptionFees` function (`FraxEtherRedemptionQueue.sol#L147-163`)

Important state parameters related to fees, such as the `feeRecipient`, are not set by the constructor of the `FraxEtherRedemptionQueue` contract. This leaves them susceptible to human error or timing issues whenever they are updated. If fees are being collected by the contract and the `feeRecipient` has not been updated from its default value of `address(0)`, the lack of checks in the `collectRedemptionFees` function will cause the collected fees to be sent to the zero address and effectively burned.

Exploit Scenario

Bob, a Frax Finance administrator, decides to enable fees on the `FraxEtherRedemptionQueue` contract. Bob submits a transaction through the timelock controller to the `setRedemptionFee` function. The `FraxEtherRedemptionQueue` contract continues operating but begins accruing fees from users. Bob submits a transaction through the timelock controller to `collectRedemptionFees`, which sends the collected amount to the zero address.

Recommendations

Short term, set the `feeRecipient` to an appropriate address in the constructor of the `FraxEtherRedemptionQueue` contract. Add a zero address check to the `feeRecipient` inside the `collectRedemptionFees` function.

Long term, expand the unit testing suite to cover more invalid or unexpected order of operations to help surface issues like this one.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Front-Running Resistance	The system's resistance to front-running attacks
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

C. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On October 18, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Frax Finance team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the five issues described in this report, Frax Finance has resolved three issues and has not resolved the remaining two. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	FRAX tokens mistakenly sent to FXB cannot be retrieved	Informational	Unresolved
2	Oracle deployed with misconfigured base token	Low	Resolved
3	Array assignment to index out of bounds breaks swap functions	High	Resolved
4	Incorrect order of arguments passed to swap function triggers revert	High	Resolved
5	collectRedemptionFees can be called before feeRecipient has been set	Medium	Unresolved

Detailed Fix Review Results

TOB-FBSRO-1: FRAX tokens mistakenly sent to FXB cannot be retrieved
Unresolved. Frax Finance did not include a mechanism to retrieve FRAX tokens that are mistakenly sent to the FXB contract. Additionally, the team has not indicated any plans to update its user-facing documentation to provide a warning about this issue.

TOB-ARCADE-2: Oracle deployed with misconfigured base token
Resolved in commit `42120ad`. The oracle contract, which was initially deployed with the wrong base token address, has been redeployed with the correct address.

TOB-ARCADE-3: Array assignment to index out of bounds breaks swap functions
Resolved in commit 42120ad. The `swapExactTokensForTokens` function has been updated to reassign the `_amounts` variable using the `new` keyword and a static array, enabling index assignment. Additionally, tests have been added to verify the expected functionality of this function.

TOB-ARCADE-4: Incorrect order of arguments passed to swap function triggers revert
Resolved in commit 42120ad. The codebase has been updated to use named arguments. In the `swapExtactTokensForTokens` method, the call to the `swap` function now has the correct order of arguments.

TOB-ARCADE-5: collectRedemptionFees can be called before feeRecipient has been set
Unresolved. The Frax Finance development team chose not to address this issue.

D. Fix Review Status Categories

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.