AladdinDAO f(x) Protocol

Security Assessment

April 16, 2024

Prepared for:
Sharlyn Wu
AladdinDAO

Prepared by: **Troy Sargent and Robert Schneider**

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.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
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:

Troy Sargent, Consultant
troy.sargent@trailofbits.com

Robert Schneider, Consultant
robert.schneider@trailofbits.com

Project Timeline

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

Date	Event
March 1, 2024	Pre-project kickoff call
March 8, 2024	Status update meeting #1
March 18, 2024	Delivery of report draft
March 18, 2024	Final readout
April 16, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

AladdinDAO engaged Trail of Bits to review the security of their f(x) protocol. It is a decentralized stablecoin system that creates two Ethereum derivative assets: fractional ETH (fETH) with low volatility and leveraged ETH (xETH) for leveraged long positions, using a mix of pure ETH and liquid staked ETH derivatives as collateral.

A team of two consultants conducted the review from March 4, 2024 to March 15, 2024, for a total of four engineer-weeks of effort. Our testing efforts focused on version two of the f(x) protocol. With full access to source code and documentation, we performed static testing and manual review of the files listed in the "Project Targets" section, using automated and manual processes.

Prior to our review, a vulnerability in the router contracts of the f(x) protocol was discovered and exploited. We did not report the vulnerability, as it was reportedly fixed by the developers in a PR following the target of our review (our coverage was limited to files added or updated in PR#162).

Observations and Impact

Our review of the protocol revealed that the codebase has grown organically without being refactored for simplicity or having its design specified, as evidenced by the complex architecture, inconsistent data validation, and untested assumptions that enable undefined behavior. We identified discrepancies between the protocol's documentation and the current implementation as well as many undiscussed scenarios, making it difficult to determine if the implementation matches the expected behavior (TOB-ADFX-5, TOB-ADFX-8, TOB-ADFX-19). Additionally, the system lacks automated testing of system properties and would greatly benefit from implementing fuzz testing (TOB-ADFX-17, TOB-ADFX19, TOB-ADFX-20). Admin roles have control over system parameters and are not controlled by a governance contract or subject to a waiting period in a timelock contract (TOB-ADFX-2, TOB-ADFX-13). Lastly, it is unclear whether the incentives for users are compatible with the health of the protocol and whether the system's risk tolerance adequately mitigates against delinquency (TOB-ADFX-16, TOB-ADFX-18, TOB-ADFX19, TOB-ADFX-19, TOB-ADFX-20).

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that AladdinDAO 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.

- **Update documentation to reflect V2 updates.** The new version of the f(x) protocol introduces new features and alters the expected behavior, which should be accurately described in the documentation. These changes may require additional guidance on proper use and potential risks.
- **Identify system invariants and fuzz test them.** The system is currently underspecified and has some areas, particularly related to stability and incentives (TOB-ADFX–16, TOB-ADFX-18, TOB-ADFX-20), that should be researched further. Explicitly listing "what is a bug" and codifying it in automated testing will proactively identify vulnerabilities and prevent regressions.
- **Consolidate the current version of the protocol into its own repository and separate the documentation pages.** Older versions of the protocol are intermixed with the new version of the protocol in the current repository, and the existing documentation contains outdated explanations and references. This will ease the maintenance burden, reduce the complexity of fixing bugs and adding new features, and make triaging security issues more manageable.
- **Simplify the architecture and reduce the number of components.** Important invariants checks (e.g., the protocol's collateralization) are performed sporadically throughout the system instead of where they are most relevant (TOB-ADFX-1, TOB-ADFX-20). This sprawl makes modifying the code without introducing a bug and reviewing the code for correctness more difficult. Instead of designing the protocol to be flexible and permissive, we recommend making it simple and strict.

Finding Severities and Categories

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

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	4
Low	8
Informational	7
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Configuration	2
Data Validation	6
Denial of Service	2
Patching	1
Undefined Behavior	9

Project Goals

The engagement was scoped to provide a security assessment of the AladdinDAO f(x) Protocol. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are access controls appropriately configured for system components?
- Can the system be manipulated through transaction front-running?
- Is there a risk of participants misappropriating or losing tokens?
- Are the anticipated actions of users in stability mode incentive compatible?
- Could arithmetic overflows compromise the system's integrity?
- Are there denial-of-service vulnerabilities that halt the system or prevent users from taking some action?
- How would flash crashes or oracle malfunctions affect the system's stability?
- Are calculations related to liquidations and the management of collateral pools accurate and reliable?
- Can users avoid losses from liquidations?
- Are fractional token (fToken) holders able to reclaim their portion of the collateral in worse case scenarios?

Project Targets

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

f(x) and fxUSD

Repository	https://github.com/AladdinDAO/aladdin-v3-contracts
Version	a2597e14fbfe6c0f2075869276aa86bf481c4dbd
Type	Solidity
Platform	Ethereum

Files

- contracts/f(x)/math/FxStableMath.sol
- contracts/f(x)/oracle/FxFrxETHTwapOracle.sol
- contracts/f(x)/rate-provider/ERC4626RateProvider.sol
- contracts/f(x)/rate-provider/WstETHRateProvider.sol
- contracts/f(x)/rebalance-pool/FxUSDShareableRebalancePool.sol
- contracts/f(x)/rebalancer/FxUSDRebalancer.sol
- contracts/f(x)/reserve-pool/ReservePoolV2.sol
- contracts/f(x)/v2/FractionalTokenV2.sol
- contracts/f(x)/v2/FxInitialFund.sol
- contracts/f(x)/v2/FxUSD.sol
- contracts/f(x)/v2/LeveragedTokenV2.sol
- contracts/f(x)/v2/MarketV2.sol
- contracts/f(x)/v2/TreasuryV2.sol
- contracts/f(x)/v2/WrappedTokenTreasuryV2.sol
- contracts/f(x)/wrapper/LeveragedTokenWrapper.sol
- contracts/gateways/facets/FxMarketV1Facet.sol
- contracts/gateways/facets/FxUSDFacet.sol
- contracts/gateways/facets/TokenConvertManagementFacet.sol
- contracts/gateways/libraries/LibGatewayRouter.sol
- contracts/voting-escrow/gauges/GaugeControllerOwner.sol

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:

- **Documentation review.** We examined the publicly available documentation for the f(x) protocol. However, this documentation pertains to a previous version of the system and only partially applies to the contracts in scope for our review. We also looked at in-line code comments and NatSpec comments to assess whether the system conforms to what is specified.
- **Static analysis.** We ran Slither on the codebase and triaged the results relevant to the in-scope contracts.
- **Testing coverage.** We ran the test suite and code assessed the coverage for gaps. We considered only tests that were covering files that were in scope.
- **Manual review.** We manually reviewed the contract's code for security vulnerabilities, logical errors, and adherence to best practices.
    - We assessed the reliability and security of the price oracles used to determine collateral valuation and Ethereum's real world price. We also evaluated the protocol's mechanisms for managing oracle failures or price manipulation.
    - We evaluated the liquidation process' design and implementation in terms of fairness, transparency, and effectiveness. The clarity of the triggers and procedures for liquidation, as well as their alignment with user and protocol interests, was also examined.
    - We evaluated the protocol's risk management strategies, focusing on its handling of extreme market conditions, liquidation cascades, and systemic risks.
    - We examined the protocol's management of debt related to leveraged positions, token redemptions, and the handling of underwater positions.
    - We assessed whether the protocol adequately tracks a user's share of the collateral and assessed points of value transfer for access control, state management, proper valuation, and correct liquidation thresholds.
    - We reviewed the role-based access control implementations and assessed how administrative decisions could affect critical system state parameters.

- We examined how rewards from staked collateral are calculated, distributed, and claimed by users.

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:

- We did not review the reward calculation library borrowed from the Liquity protocol. Additionally, we did not perform a differential analysis comparing its implementation in Liquity with its use in the f(x) protocol.
- We were informed that the “strategies” concept used by the TreasuryV2 contract is not fully implemented. Therefore, we treated this as a black box.
- We did not conduct an in-depth review of third-party integrations such as Curve, Balancer, Chainlink, Lido, and Frax. Undefined behaviors or corner cases that might arise from interactions with these external systems should be further investigated.
- We did not exhaustively review the economic sustainability of the protocol, particularly in terms of how system parameters, such as collateralization ratios, reward distribution mechanisms, and leverage limits, could affect its long-term viability.
- A fix for a vulnerability in the gateway contracts was implemented just before our review began. These fixes were not included in the scope of this audit and therefore were not reviewed.
- We did not conduct a detailed review of the `chainlinkETHTwapOracle` or the `curvePool` contracts, which both feed price data to the `FxFrxETHTwapOracle` contract. We limited our analysis to the `FxFrxETHTwapOracle` contract, as it was the only one in scope.

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	Although the system's use of unchecked arithmetic is justified, none of the calculations explicitly specify the reasoning for rounding up or down (TOB-ADFX-17). The system configurations are not consistently bounded (TOB-ADFX-13).	**Moderate**
Auditing	Events are adequately implemented for movement of assets and state variable updates. The protocol team should identify scenarios that warrant pausing the system and document who is responsible to monitor and intervene. This should inform the creation of a full-fledged incident response plan and rehearsals.	**Moderate**
Authentication / Access Controls	While the roles in core contracts are relatively straightforward, the protocol lacks system diagrams of roles, documentation on their responsibilities, and information on their key management.	**Moderate**
Complexity Management	The system should be simplified both in its scope of features and implementation with a focus on creating logical boundaries for the components. Currently, the contracts do not have separation of concerns for their logic (TOB-ADFX-20). Validations are not performed in the component they are relevant to; instead, validation is outsourced to separate components. Additionally, some actions require interacting with many cross-contract invocations and could likely be simplified with a different architecture. The economic incentives of the system are complex, and it is not always clear that the expected actions of users are incentive compatible (TOB-ADFX-16, TOB-ADFX-19).	**Weak**

	These incentives should be investigated further, inform design changes, and be documented.	
Decentralization	Admin roles are held by externally owned accounts or multisig contracts, and there are not any timelocks. Users do not have adequate time to respond to changes to the system configuration, like the stability ratio and liquidation ratio (TOB-ADFX-2). The admin can move collateral out of the system that may not be recovered without permission (TOB-ADFX-5).	**Weak**
Documentation	While the function- and contract-level documentation is adequate, the complex interactions and scenarios that the protocol may enter are not adequately documented. The documentation is out of date and contains references that are relevant only to prior versions of the protocol. We recommend creating a state machine representation of the system and using it to inform the architecture and validation of components. Additionally, creating diagrams that show the payoff amount that users should expect to receive in a given scenario would help clarify risks.	**Moderate**
Low-Level Manipulation	The contracts in scope use assembly code minimally, mostly in peripheral code such as libraries.	**Not Applicable**
Testing and Verification	The protocol would benefit from end-to-end scenarios and integration tests, particularly surrounding liquidations and redemptions of tokens in worst-case scenarios. While there are unit tests for complex math formulas, they test for few inputs and could benefit from more exhaustive testing. Most importantly, the properties of the system that should never break should be identified and created, and they should be converted to fuzz tests to gain assurance that they hold.	**Moderate**
Transaction Ordering	The liquidation system is first-come first-serve, but the complexity of the auction model may not be worthwhile. While it is purposeful, allowing users to withdraw from the rebalance pool during stability mode by front-running liquidations may not promote	**Moderate**

	re-collateralization (TOB-ADFX-19).	

Summary of Findings

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

ID	Title	Type	Severity
1	Redeeming xToken and fToken simultaneously uses incorrect TWAP price	Undefined Behavior	Informational
2	liquidatableCollateralRatio update can force liquidation without warning	Configuration	Low
3	Panicking checked arithmetic may prevent deposits and withdraws to rebalance pool	Denial of Service	Low
4	Incorrect TWAP price may be used for calculations such as collateral ratio	Undefined Behavior	Medium
5	Updating strategy may cause users to lose funds during redemption	Undefined Behavior	Medium
6	Time-weighted Chainlink oracle can report inaccurate price	Configuration	Medium
7	Net asset value of fractional and leverage token may reflect invalid price	Data Validation	Low
8	Collateral ratio does not account underlying value of collateral in strategy	Undefined Behavior	Informational
9	Rewards are withdrawn even if protocol is not sufficiently collateralized	Undefined Behavior	Informational
10	Rebalance pool withdrawal silently fails	Undefined Behavior	Low
11	Upgradeable contract initialization calls are commented out	Patching	Informational
12	Sum of all user shares does not equal total supply	Data Validation	Low

13	Lack of validation when updating system configurations	Data Validation	**Low**
14	Lack of slippage checks prevents user from specifying acceptable loss	Data Validation	**Low**
15	Deployments to L2 should check sequencer uptime for Chainlink price feeds	Data Validation	**Low**
16	Treating fToken as $1 creates arbitrage opportunity and unclear incentives	Undefined Behavior	**Informational**
17	Rounding direction for deposits does not favor the protocol	Undefined Behavior	**Undetermined**
18	Reverting when minting xToken can prevent re-collateralization	Denial of Service	**Medium**
19	Unclear economic sustainability of allowing user to avoid liquidations	Undefined Behavior	**Informational**
20	Validation of system invariants is error prone	Data Validation	**Informational**

Detailed Findings

1. Redeeming xToken and fToken simultaneously uses incorrect TWAP price

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ADFX-1
Target: `TreasuryV2.sol`	

Description

The `Market` contract redeems either the leverage token (xToken) or fractional token (fToken) and is currently the only contract that can redeem tokens from the Treasury contract. However, the market role can be granted to contracts, which permits simultaneous redemption of both token types. If both tokens are redeemed at once, the xToken will receive the same price as if only an fToken were being redeemed. Following an update, a user may be able to manipulate the price to be `maxPrice` by depositing just a single `fToken` and consequently receive a greater amount of the base asset.

```
function redeem(
  uint256 _fTokenIn,
  uint256 _xTokenIn,
  address _owner
) external override onlyRole(FX_MARKET_ROLE) returns (uint256 _baseOut) {
  FxStableMath.SwapState memory _state;
  if (_fTokenIn > 0) {
    _state = _loadSwapState(Action.RedeemFToken);
  } else {
    _state = _loadSwapState(Action.RedeemXToken);
  }
[...]
  if (_state.xNav == 0) {
    if (_xTokenIn > 0) revert ErrorUnderCollateral();
    // only redeem fToken proportionallly when under collateral.
    _baseOut = (_fTokenIn * _state.baseSupply) / _state.fSupply;
  } else {
    _baseOut = _state.redeem(_fTokenIn, _xTokenIn);
  }
[...]
  _transferBaseToken(_baseOut, msg.sender);
}
```

Figure 1.1: Implementation of the Treasury's `redeem` function (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#324–357)

```
function _loadSwapState(Action _action) internal view returns
(FxStableMath.SwapState memory _state) {
  _state.baseSupply = totalBaseToken;
  _state.baseNav = _fetchTwapPrice(_action);
```

Figure 1.2: The base assets value is computed using `_loadSwapState` (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#607–609)

Currently, redeeming an xToken will use the `minPrice` from the TWAP (time-weighted average price) to calculate how much base asset the user will receive. However, the `_fetchTwapPrice` function does not handle how the price should be determined in cases where both assets are being redeemed at once. As such, the Treasury should prevent this.

```
function _fetchTwapPrice(Action _action) internal view returns (uint256 _twapPrice)
{
  (bool _isValid, uint256 _safePrice, uint256 _minPrice, uint256 _maxPrice) =
IFxPriceOracle(priceOracle).getPrice();

  _twapPrice = _safePrice;
  if (_action == Action.MintFToken || _action == Action.MintXToken) {
    if (!_isValid) revert ErrorInvalidOraclePrice();
  } else if (!_isValid) {
    if (_action == Action.RedeemFToken) {
      _twapPrice = _maxPrice;
    } else if (_action == Action.RedeemXToken) {
      _twapPrice = _minPrice;
    }
  }
```

Figure 1.3: `_fetchTwapPrice` expects only one action (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#644–656)

Exploit Scenario

A new market role is granted to a contract that allows redeeming xToken and fToken simultaneously, allowing the xToken to be redeemed at the incorrect price.

Recommendations

Short term, require that only one token is redeemed at a time by adding additional validation or creating two distinct entry points for redeeming either token.

Long term, strictly validate inputs from external contracts (even trusted) to ensure upgrades or configuration do not allow an important check to be bypassed.

2. liquidatableCollateralRatio update can force liquidation without warning

Severity: **Low**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-ADFX-2
Target: `BoostableRebalancePool.sol`	

Description

The `DEFAULT_ADMIN_ROLE` can update the `liquidatableCollateralRatio` without notifying users who hold positions in the rebalance pool. This action could potentially trigger or hinder liquidations without giving users a chance to make adjustments.

Only the `DEFAULT_ADMIN_ROLE`, which is a multisig wallet address controlled by the AladdinDAO development team, can call the update function for this important system parameter. This implies that updates will take effect immediately, without prior notice to protocol users.

```
 function updateLiquidatableCollateralRatio(uint256 _newRatio) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    uint256 _oldRatio = liquidatableCollateralRatio;
    liquidatableCollateralRatio = _newRatio;

    emit UpdateLiquidatableCollateralRatio(_oldRatio, _newRatio);
 }
```

Figure 2.1: The `updateLiquidatableCollateralRatio` function (`BoostableRebalancePool.sol#376–381`)

The `liquidatableCollateralRatio` is used in the `liquidate()` function to benchmark the system's health and determine if a liquidation is required.

```
 if (_treasury.collateralRatio() >= liquidatableCollateralRatio) {
   revert CannotLiquidate();
 }
 (, uint256 _maxLiquidatable) =
_treasury.maxRedeemableFToken(liquidatableCollateralRatio);
```

Figure 2.1: The `updateLiquidatableCollateralRatio` function (`BoostableRebalancePool.sol#313–316`)

If the `liquidatableCollateralRatio` were raised while the `_treasury.collateralRatio()` stays the same, a system in good standing could suddenly change so that the `liquidate` function can be successfully run.

Exploit Scenario

Bob, an f(x) protocol user, has a leveraged position of xETH that is close to the liquidation threshold. The AladdinDAO team calls `updateLiquidatableCollateralRatio()` and raises the `liquidatableCollateralRatio` variable. The `LIQUIDATOR_ROLE` calls `liquidate()` and Bob, along with other users' positions, lose value.

Recommendations

Short term, give access control to a role that is managed by a governance contract where system parameter changes are subject to a timelock. Enforce a minimum and maximum bound for the `liquidatableCollateralRatio` to provide users with a clear expectation of any potential updates.

Long term, when important protocol parameters are updated, it is crucial to allow users to adjust their positions or exit the system before these updates take effect.

3. Panicking checked arithmetic may prevent deposits and withdraws to rebalance pool

Severity: **Low**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-ADFX-3
Target: `ShareableRebalancePool.sol`, `ShareableRebalancePoolV2.sol`	

Description

The rebalance pool's deposit and withdraw functionality computes the latest balance of a user's account at each interaction. This calculation strictly assumes that the `_newBalance` amount is less than the old, but does not validate that this is the case. If this is ever not the case, the pool will revert due to Solidity's built-in checked arithmetic, and the user will not be able to deposit or withdraw.

```
function deposit(uint256 _amount, address _receiver) external override {
[...]
  _checkpoint(_receiver);
```

Figure 3.1: Deposit invokes `_checkpoint` (aladdin-v3-contracts/contracts/f(x)/rebalance-pool/ShareableRebalancePool.sol#250–263)

```
function _withdraw(
  address _sender,
  uint256 _amount,
  address _receiver
) internal {
  _checkpoint(_sender);
```

Figure 3.2: `_withdraw` invokes `_checkpoint` (aladdin-v3-contractsaladdin-v3-contracts/contracts/f(x)/rebalance-pool/ShareableRebalancePool.sol#539–545)

```
function _checkpoint(address _account) internal virtual override {
[...]
    TokenBalance memory _balance = _updateUserBalance(_account, _supply);
    TokenBalance memory _ownerBalance = _updateVoteOwnerBalance(_owner, _supply);
    _updateBoostCheckpoint(_account, _owner, _balance, _ownerBalance, _supply);
  }
}
```

Figure 3.3: `_checkpoint` invokes `_updateUserBalance` (aladdin-v3-contracts/contracts/f(x)/rebalance-pool/ShareableRebalancePool.sol#465–488)

The following check should resemble `_balance.amount > _newBalance` in order to avoid reverting on the overflow case and emit an event only when a real loss occurs (i.e., the user's balance has decreased as indicated by the `UserDepositChange` event's `loss` parameter).

```
uint104 _newBalance = uint104(_getCompoundedBalance(_balance.amount,
_balance.product, _supply.product));
if (_newBalance != _balance.amount) {
  // no unchecked here, just in case
  emit UserDepositChange(_account, _newBalance, _balance.amount - _newBalance);
}
```

Figure 3.4: `_updateUserBalance` can revert and prevent deposits/withdrawals (aladdin-v3-contracts/contracts/f(x)/rebalance-pool/ShareableRebalancePool.sol#649–653)

Exploit Scenario

A user deposits to the rebalance pool and is unable to withdraw because an overflow causes the transaction to revert.

Recommendations

Short term, validate that `_balance.amount > _newBalance` is true before unconditionally performing checked arithmetic during interactions that users must be able to successfully complete.

Long term, review the logical and arithmetic operations for incorrect assumptions and perform fuzz testing to identify corner cases like these.

4. Incorrect TWAP price may be used for calculations such as collateral ratio

Severity: **Medium**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-ADFX-4
Target: `TreasuryV2.sol`	

Description

The swap state of the protocol uses the enum value, `Action.None` when it is loaded for operations aside from minting and redeeming. The implementation of `_fetchTwapPrice` does not revert when the action is `Action.None` and the TWAP price is invalid, allowing the invalid price to be used to compute the collateral ratio. Since other contracts rely on these values to perform important validations and calculations, it is important that the price is valid and accurate.

```
function collateralRatio() public view override returns (uint256) {
  FxStableMath.SwapState memory _state = _loadSwapState(Action.None);

  if (_state.baseSupply == 0) return PRECISION;
  if (_state.fSupply == 0) return PRECISION * PRECISION;

  return (_state.baseSupply * _state.baseNav) / _state.fSupply;
}
```

Figure 4.1: `collateralRatio` uses `Action.None` for its swap state (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#177–184)

```
function _fetchTwapPrice(Action _action) internal view returns (uint256 _twapPrice)
{
  (bool _isValid, uint256 _safePrice, uint256 _minPrice, uint256 _maxPrice) =
IFxPriceOracle(priceOracle).getPrice();

  _twapPrice = _safePrice;
  if (_action == Action.MintFToken || _action == Action.MintXToken) {
    if (!_isValid) revert ErrorInvalidOraclePrice();
  } else if (!_isValid) {
    if (_action == Action.RedeemFToken) {
      _twapPrice = _maxPrice;
    } else if (_action == Action.RedeemXToken) {
      _twapPrice = _minPrice;
    }
  }
  if (_twapPrice == 0) revert ErrorInvalidTwapPrice();
}
```

Figure 4.2: `_fetchTwapPrice` does handle an invalid price for `Action.None` (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#644–659)

This also may affect the following calculations, `isUnderCollateral` and `currentBaseTokenPrice`, and other operations that use `_loadSwapState(Action.None)`.

```
function isUnderCollateral() external view returns (bool) {
  FxStableMath.SwapState memory _state = _loadSwapState(Action.None);
  return _state.xNav == 0;
}
```

Figure 4.3: `isUnderCollateral` may use invalid price (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#187–190)

```
function currentBaseTokenPrice() external view override returns (uint256) {
  return _fetchTwapPrice(Action.None);
}
```

Figure 4.4: `currentBaseTokenPrice` may use invalid price (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#249–251)

Exploit Scenario

The rebalancer pool contract permits liquidations because it uses a value of `collateralRatio` that is based on an invalid price. This causes users to be liquidated at an unfair price.

Recommendations

Short term, define and implement what price should be used if the price is invalid when the action is `Action.None` or revert, if appropriate.

Long term, ensure the integrity and correctness of price calculations, especially those related to minting/ burning and depositing/withdrawing.

5. Updating strategy may cause users to lose funds during redemption

Severity: **Medium**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ADFX-5
Target: `TreasuryV2.sol`	

Description

Collateral in the Treasury contract may be deposited into a strategy contract that the admin can update at any time. Since the update does not prevent collateral from being left in the old strategy contract or reset the balance that tracks the amount deposited in the `strategyUnderlying` strategy, users may receive substantially less than they are owed upon redeeming xToken and fToken, even though the collateral is available in the old strategy contract. Note that users may specify a minimum amount out in the market contract, but an admin would have to intervene and handle recovering the strategy's assets to make them available to the user.

```
function updateStrategy(address _strategy) external onlyRole(DEFAULT_ADMIN_ROLE) {
  _updateStrategy(_strategy);
}
```

Figure 5.1: Admin can update strategy without any validation (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#452–455)

```
function redeem(
  uint256 _fTokenIn,
  uint256 _xTokenIn,
  address _owner
) external override onlyRole(FX_MARKET_ROLE) returns (uint256 _baseOut) {
[...]
  if (_fTokenIn > 0) {
    IFxFractionalTokenV2(fToken).burn(_owner, _fTokenIn);
  }
  if (_xTokenIn > 0) {
    IFxLeveragedTokenV2(xToken).burn(_owner, _xTokenIn);
  }

  totalBaseToken = _state.baseSupply - _baseOut;

  _transferBaseToken(_baseOut, msg.sender);
}
```

Figure 5.2: The `redeem` function invokes `_transferBaseToken` (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#324–357)

The implementation of `_transferBaseToken` overrides the amount withdrawn when the strategy fails to return the remainder of the tokens owed to the user. Because `strategyUnderlying` may be the balance deposited to previous strategies, it is possible that the current strategy does not have sufficient balance to fulfill the deficit; however, the operation, `strategyUnderlying` minus `_diff`, will not overflow. This may cause the user to receive substantially less because the balances of the strategy contracts are not fully withdrawn before an admin updates the strategy contract.

```
function _transferBaseToken(uint256 _amount, address _recipient) internal returns
(uint256) {
  _amount = getWrapppedValue(_amount);

  uint256 _balance = IERC20Upgradeable(baseToken).balanceOf(address(this));
  if (_balance < _amount) {
    uint256 _diff = _amount - _balance;
    IAssetStrategy(strategy).withdrawToTreasury(_diff);
    strategyUnderlying = strategyUnderlying - _diff;

    // consider possible slippage here.
    _balance = IERC20Upgradeable(baseToken).balanceOf(address(this));
    if (_amount > _balance) {
      _amount = _balance;
    }
  }

  IERC20Upgradeable(baseToken).safeTransfer(_recipient, _amount);
```

Figure 5.3: Users may receive less than expected if the strategy does return the full deficit. (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#585–601)

Exploit Scenario

The admin updates the strategy while collateral is deposited. As a result, `strategyUnderlying`—the number of base tokens in the strategy—is greater than zero. If a user redeems via the `Market` contract and the Treasury cannot successfully withdraw sufficient funds from the new strategy, the users' fToken and xToken will have been burned without receiving their share of the base asset.

Recommendations

Short term, prior to updating the strategy, ensure that all funds have been withdrawn into the Treasury and that `strategyUnderlying` is reset to 0.

Long term, investigate how to handle scenarios where the available collateral declines due to losses incurred by the strategy.

6. Time-weighted Chainlink oracle can report inaccurate price

Severity: **Medium**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-ADFX-6
Target: `contracts/price-oracle/twap/ChainlinkTwapOracleV3.sol`	

Description
The Chainlink oracle used to price the value of ETH in USD is subject to a time-weighted average price that will over-represent or under-represent the true price in times of swift volatility. This can increase the likelihood of bad debt if a sudden price change is not acted on due to the lag, and it is too late once the collateral's loss in value is reflected in the TWAP price.

```
function _fetchPrice() internal view returns (CachedPrice memory _cached) {
  _cached.ETH_USDPrice =
ITwapOracle(chainlinkETHTwapOracle).getTwap(block.timestamp);
  _cached.frxETH_ETHPrice = ICurvePoolOracle(curvePool).ema_price();
  _cached.frxETH_USDPrice = (_cached.ETH_USDPrice * _cached.frxETH_ETHPrice) /
PRECISION;
}
```

Figure 6.1: Use of TWAP computed from Chainlink feed (aladdin-v3-contracts/contracts/f(x)/oracle/FxFrxETHTwapOracle.sol#85–89)

Exploit Scenario
The price of Ethereum jumps 10% over the course of 10 minutes. Eve notices the price discrepancy between the actual price and the reported price, and takes out a 4x long position on the frxETH pool. Once the time-weighted average price of ETH catches up to the actual price, Eve cashes out her long position for a profit.

Recommendations
Short term, remove the reliance on a time-weighted average price, at least for liquidations. It is best to be punitive when pricing debt and conservative when valuing collateral, so the TWAP may be appropriate for minting.

Long term, adhere to best practices for oracle solutions and ensure backup oracles and safety checks do not create credit risk.

7. Net asset value of fractional and leverage token may reflect invalid price

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-ADFX-7
Target: `contracts/f(x)/v2/LeveragedTokenV2.sol`, `contracts/f(x)/v2/FractionalTokenV2.sol`	

Description

The contracts for the fToken and xToken minted by the Treasury expose a method, `nav`, to retrieve the net asset value (NAV) of tokens. This represents what portion of Treasury's reserve collateral is available to be redeemed in exchange for the given token. However, the NAV uses `currentBaseTokenPrice` indiscriminately and fails to validate that the price is valid. As outlined in the aforementioned issue (TOB-ADFX-4), fetching the TWAP using the `Action.None` value may return an invalid price. External integrations that rely on this price may misreport the true NAV or use the incorrect value in calculations, and will therefore get incorrect results.

```
function currentBaseTokenPrice() external view override returns (uint256) {
  return _fetchTwapPrice(Action.None);
}
```

Figure 7.1: Base token price may return invalid TWAP price (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#249–251)

```
function nav() external view override returns (uint256) {
  uint256 _xSupply = totalSupply();
  if (IFxTreasuryV2(treasury).isUnderCollateral()) {
    return 0;
  } else if (_xSupply == 0) {
    return PRECISION;
  } else {
    uint256 baseNav = IFxTreasuryV2(treasury).currentBaseTokenPrice();
    uint256 baseSupply = IFxTreasuryV2(treasury).totalBaseToken();
    uint256 fSupply = IERC20Upgradeable(fToken).totalSupply();
    return (baseNav * baseSupply - fSupply * PRECISION) / _xSupply;
  }
}
```

Figure 7.2: Leveraged token uses `currentBaseTokenPrice` without checking it is valid (aladdin-v3-contracts/contracts/f(x)/v2/LeveragedTokenV2.sol#55–67)

```
function nav() external view override returns (uint256) {
  uint256 _fSupply = totalSupply();
  if (_fSupply > 0 && IFxTreasuryV2(treasury).isUnderCollateral()) {
    // under collateral
    uint256 baseNav = IFxTreasuryV2(treasury).currentBaseTokenPrice();
    uint256 baseSupply = IFxTreasuryV2(treasury).totalBaseToken();
    return (baseNav * baseSupply) / _fSupply;
  } else {
    return PRECISION;
  }
}
```

Figure 7.3: Fractional token uses `currentBaseTokenPrice` without checking it is valid (`aladdin-v3-contracts/contracts/f(x)/v2/FractionalTokenV2.sol#53–63`)

Recommendations

Short term, consider reverting if the price is deemed invalid rather than reflecting the invalid price in the net asset value, or add an additional return value that external integrations can use to determine whether the NAV is based on an invalid price.

Long term, perform consistent validation throughout the codebase and add tests for scenarios where the price is considered invalid.

8. Collateral ratio does not account underlying value of collateral in strategy

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ADFX-8
Target: `contracts/f(x)/v2/TreasuryV2.sol`	

Description

When collateral is moved into the strategy, the `totalBaseToken` amount remains unchanged. This does not account for shortfalls and surpluses in the value of the collateral moved into the strategy, and as a result, the collateral ratio may be over-reported or under-reported, respectively.

The integration of strategies in the treasury has not been fully designed and has flaws as evidenced by TOB-ADFX-5. This feature should be reconsidered and thoroughly specified prior to active use. For example, it is unclear whether the strategy's impact on the collateral ratio is robust below 130%. It may benefit the stability of the system to prevent too much of the collateral from entering strategies during periods of high volatility or when the system is not overcollateralized by implementing a buffer, or a maximum threshold of the amount of collateral devoted to strategies.

```
function transferToStrategy(uint256 _amount) external override onlyStrategy {
  IERC20Upgradeable(baseToken).safeTransfer(strategy, _amount);
  strategyUnderlying += _amount;
}
```

Figure 8.1 Collateral is moved into strategy without updating `totalBaseToken` (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#375–378)

Since `totalBaseToken` is not updated for strategies, it may not be reflected correctly in the amount of base token available to be harvested as reported by the `harvestable` function.

```
function harvestable() public view returns (uint256) {
  uint256 balance = IERC20Upgradeable(baseToken).balanceOf(address(this));
  uint256 managed = getWrapppedValue(totalBaseToken);
  if (balance < managed) return 0;
  else return balance - managed;
}
```

Figure 8.2: The rewards available to harvest does not consider value of strategy (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#274–279)

Recommendations

Short term, disable the use of strategies for active deployments and consider removing the functionality altogether until it is fully specified and implemented.

Long term, limit the number of features in core contracts and do not prematurely merge incomplete features.

9. Rewards are withdrawn even if protocol is not sufficiently collateralized

Severity: **Informational**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-ADFX-9
Target: `contracts/f(x)/v2/TreasuryV2.sol`	

Description

The treasury allows any excess base token amount beyond what is considered collateral to be distributed by calling the `harvest` function, `harvest`. Some of this token amount is sent to the rebalance pool as rewards. Since the rebalance pool's purpose is to encourage the re-collateralization of treasury, it follows that temporarily withholding the rewards until the protocol is re-collateralized, or using them to cover shortfalls, may more effectively stabilize the system.

```
/// @notice Harvest pending rewards to stability pool.
function harvest() external {
  FxStableMath.SwapState memory _state = _loadSwapState(Action.None);
  _updateEMALeverageRatio(_state);

  uint256 _totalRewards = harvestable();
  uint256 _harvestBounty = (getHarvesterRatio() * _totalRewards) / FEE_PRECISION;
  uint256 _rebalancePoolRewards = (getRebalancePoolRatio() * _totalRewards) /
FEE_PRECISION;

  emit Harvest(msg.sender, _totalRewards, _rebalancePoolRewards, _harvestBounty);

  if (_harvestBounty > 0) {
    IERC20Upgradeable(baseToken).safeTransfer(_msgSender(), _harvestBounty);
    unchecked {
      _totalRewards = _totalRewards - _harvestBounty;
    }
  }

  if (_rebalancePoolRewards > 0) {
    _distributeRebalancePoolRewards(baseToken, _rebalancePoolRewards);
    unchecked {
      _totalRewards = _totalRewards - _rebalancePoolRewards;
    }
  }

  if (_totalRewards > 0) {
    IERC20Upgradeable(baseToken).safeTransfer(platform, _totalRewards);
  }
}
```

Figure 9.1: Reducing the base asset in Treasury (`aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#384–413`)

Recommendations

Short term, consider pausing reward harvesting when the collateral ratio is below the stability ratio and even using the rewards to compensate for shortfalls.

Long term, perform additional review of the incentive compatibility and economic sustainability of the system.

10. Rebalance pool withdrawal silently fails

Severity: **Low**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-ADFX-10
Target: `contracts/f(x)/rebalance-pool/FxUSDShareableRebalancePool.sol`	

Description

The `FxUSDShareableRebalancePool` contract disallows withdrawals by commenting out the internal call, which will silently fail for external integrations and users. Instead, the call should revert and provide a clear error message explaining that withdrawing fToken is disabled.

```
function withdraw(uint256 _amount, address _receiver) external override {
  // not allowed to withdraw as fToken in fxUSD.
  // _withdraw(_msgSender(), _amount, _receiver);
}
```

Figure 10.1: Withdraw function with commented-out code (aladdin-v3-contracts/contracts/f(x)/rebalance-pool/FxUSDShareableRebalancePool.sol#26–29)

Recommendations

Short term, revert with a custom error if the function, `withdraw`, is called.

Long term, do not leave commented-out code and explicitly handle error cases.

11. Upgradeable contract initialization calls are commented out

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-ADFX-11
Target: `contracts/f(x)/rebalance-pool/ShareableRebalancePool.sol`	

Description

The usage of OpenZeppelin's upgradeable smart contract library requires that the parent contracts' `init` functions are called in the child's `initialize` function. However, this is commented out in the `ShareableRebalancePool` contract. Currently, these calls are no-ops and have no effect. However, prior to upgrading the library dependency, this code should be updated to reflect the target library's implementation and ensure that the upgrade does not introduce a bug.

```
// __Context_init(); // from ContextUpgradeable, comment out to reduce codesize
// __ERC165_init(); // from ERC165Upgradeable, comment out to reduce codesize
// __AccessControl_init(); // from AccessControlUpgradeable, comment out to reduce
codesize
```

Figure 11.1: Commented-out `init` functions (aladdin-v3-contracts/contracts/f(x)/rebalance-pool/ShareableRebalancePool.sol#187–189)

Recommendations

Short term, find an alternative solution to avoid hitting the maximum code size limit.

Long term, avoid commented-out, dead code and ensure that library upgrades do not introduce new bugs.

12. Sum of all user shares does not equal total supply

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-ADFX-12
Target: `contracts/f(x)/v2/FxInitialFund.sol`	

Description

The `FxInitialFund` contract allows users to deposit up until the mint occurs, then funds can be withdrawn. Because users cannot deposit after withdrawals are enabled, the `totalSupply` is never updated and is therefore not accurate after withdrawals.

While this does not affect the pro-rata share of fToken and xToken that each user receives, it does make the total supply of shares inaccurate. Rather than recompute the proportion of tokens a share is entitled to each time and hold the total supply constant, the amount of fToken and xToken each share is worth can be cached in the `mint` function, and the total supply can be decremented each time a withdrawal is made. This accomplishes the same functionality for withdrawals and ensures that the total supply of shares is accurately tracked.

```
function withdraw(address receiver) external {
  if (!initialized) revert ErrorNotInitialized();
  if (!fxWithdrawalEnabled) revert ErrorFxWithdrawalNotEnabled();

  uint256 _share = shares[_msgSender()];
  shares[_msgSender()] = 0;
  uint256 _totalShares = totalShares;
  uint256 _fAmount = (_share * totalFToken) / _totalShares;
  uint256 _xAmount = (_share * totalXToken) / _totalShares;

  IERC20(fxUSD).safeTransfer(receiver, _fAmount);
  IERC20(xToken).safeTransfer(receiver, _xAmount);
}
```

Figure 12.1: User share is set to zero but `totalSupply` is unchanged (aladdin-v3-contracts/contracts/f(x)/v2/FxInitialFund.sol#141–154)

```
function mint() external onlyRole(MINTER_ROLE) {
  if (initialized) revert ErrorInitialized();

  uint256 _balance = IERC20(baseToken).balanceOf(address(this));
  IERC20(baseToken).safeTransfer(treasury, _balance);
  (uint256 _totalFToken, uint256 _totalXToken) =
IFxTreasuryV2(treasury).initializeProtocol(
```

```
    IFxTreasuryV2(treasury).getUnderlyingValue(_balance)
  );

  IERC20(fToken).safeApprove(fxUSD, _totalFToken);
  IFxUSD(fxUSD).wrap(baseToken, _totalFToken, address(this));

  totalFToken = _totalFToken;
  totalXToken = _totalXToken;
  initialized = true;
}
```

Figure 12.2: The `mint` function (aladdin-v3-contracts/contracts/f(x)/v2/FxInitialFund.sol#160–175)

Recommendations

Short term, compute the amount of fToken and xToken to distribute per share in the `mint` function, and update the `withdraw` function to use this amount and decrement the total supply of shares.

Long term, ensure that user balances and the sum of all user balances, total supply, is synchronized, and implement invariant testing. This can also be applied to other balances, such as rewards per individual and total available rewards.

13. Lack of validation when updating system configurations

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-ADFX-13
Target: `contracts/f(x)/v2/MarketV2.sol`	

Description

The market contract's Boolean configurations do not validate that the configuration has changed when they are updated. While setting the same value is benign, it may obscure a logical error in a peripheral program that would be readily identified if the update reverts and raises an alarm.

```
function _updateBoolInMarketConfigData(uint256 offset, bool newValue) private
returns (bool oldValue) {
  bytes32 _data = marketConfigData;
  oldValue = _data.decodeBool(offset);
  marketConfigData = _data.insertBool(newValue, offset);
}
```

Figure 13.1: Boolean config update may have no effect (aladdin-v3-contracts/contracts/f(x)/v2/MarketV2.sol#542–546)

While the stability ratio cannot be too large, the `_updateStabilityRatio` function does not validate that the stability ratio is greater than 100% (1e18), which would allow the protocol to be under-collateralized.

```
function _updateStabilityRatio(uint256 _newRatio) private {
  if (_newRatio > type(uint64).max) revert ErrorStabilityRatioTooLarge();

  bytes32 _data = marketConfigData;
  uint256 _oldRatio = _data.decodeUint(STABILITY_RATIO_OFFSET, 64);
  marketConfigData = _data.insertUint(_newRatio, STABILITY_RATIO_OFFSET, 64);

  emit UpdateStabilityRatio(_oldRatio, _newRatio);
}
```

Figure 13.2: Stability ratio does not require $1 backing for fToken (aladdin-v3-contracts/contracts/f(x)/v2/MarketV2.sol#550–558)

Recommendations

Short term, require that the new value is not equal to the old and that the stability ratio is not less than 100%.

Long term, validate that system parameters are within sane bounds and that updates are not no-ops.

14. Lack of slippage checks prevents user from specifying acceptable loss

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ADFX-14
Target: `contracts/gateways/facets/FxUSDFacet.sol`	

Description

The `router` contract to interact with the FxUSD contract and `Market` contract does not allow specifying slippage limits for redeeming xToken and mint FxUSD, respectively. Note, there is a slippage check for the target asset in the Facet contract but not for the intermediary assets exchanged during the multi-leg "swap".

```
uint256 _baseOut = IFxMarketV2(_market).redeemXToken(_amountIn, address(this), 0);
```

Figure 14.1: Lack of slippage check on redeeming xToken (aladdin-v3-contracts/contracts/gateways/facets/FxUSDFacet.sol#205)

```
uint256 _fxUSDMinted = IFxUSD(fxUSD).mint(_baseTokenIn, _amountIn, address(this),
0);
```

Figure 14.2: Lack of slippage check on minting FxUSD (aladdin-v3-contracts/contracts/gateways/facets/FxUSDFacet.sol#355)

This also affects the Balancer wrapper contract, which was not in scope of our review.

```
function unwrap(uint256 _amount) external override returns (uint256) {
  address[] memory _assets = new address[](2);
  uint256[] memory _amounts = new uint256[](2);
  _assets[srcIndex] = src;
  _assets[1 - srcIndex] = WETH;

  uint256 _balance = IERC20(src).balanceOf(msg.sender);
  IBalancerVault(BALANCER_VAULT).exitPool(
    poolId,
    address(this),
    msg.sender,
    IBalancerVault.ExitPoolRequest({
      assets: _assets,
      minAmountsOut: _amounts,
```

Figure 14.3: Lack of slippage check on Balancer swap (`aladdin-v3-contracts/contracts/f(x)/wrapper/FxTokenBalancerV2Wrapper.sol#89–102`)

Recommendations

Short term, allow users to specify a slippage tolerance for all paths.

Long term, do not hard-code arguments that users should be able to input, especially ones that protect against losses.

15. Deployments to L2 should check sequencer uptime for Chainlink price feeds

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-ADFX-15
Target: `price feed`	

Description
The protocol's current usage of Chainlink price feeds does not consider whether the sequencer is down for deployments to layer 2 blockchains (L2s) such as Arbitrum and Optimism. This can result in the usage of stale price info and unfairly impact users. For example, it may be appropriate to allot a grace period for xToken holders before allowing them to redeem their proportion of the base collateral. However, insolvent~~deeply underwater~~ treasuries should likely still be liquidated to avoid further losses.

Exploit Scenario
Holders of xToken are attempting to re-collateralize the system to prevent their NAV from becoming $0, but the L2's sequencer is down. An outdated price is used and fToken holders begin redeeming their portion of the treasury's collateral without sufficient time for xToken holders to respond once the sequencer is up.

Recommendations
Short term, implement functionality to check the sequencer uptime with Chainlink oracles for deploying the f(x) protocol to L2s.

Long term, validate that oracle prices are sufficiently fresh and not manipulated.

References
L2 Sequencer Uptime Feeds

16. Treating fToken as $1 creates arbitrage opportunity and unclear incentives

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ADFX-16
Target: `contracts/f(x)/v2/TreasuryV2.sol`	

Description

The Treasury contract treats fToken as $1 regardless of the price on reference markets. This means that anyone can buy fToken and immediately redeem for profit, and the protocol's collateralization will decrease more than it would have if valued at market value.

During periods where the protocol's collateral ratio is less than the stability ratio and fToken is valued at a premium, users may not readily redeem the fToken as expected because they will receive only $1 in collateral. This may delay or prevent re-collateralization, as minting fToken is one action that is anticipated to aid re-collateralization during stability mode.

The arbitrage opportunity may exacerbate issues such as TOB-ADFX-20 by making it profitable to purchase fToken on decentralized exchanges and redeem it in excess, but this requires further investigation.

```
uint256 _fVal = _state.fSupply * PRECISION;
```

Figure 16.1: Treasury values fToken as $1 (1e18) (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#621)

Recommendations

Short term, implement monitoring for the price of fToken and unusual activity, and create an action in an incident response plan for these scenarios.

Long term, conduct further analysis and determine when it is favorable to the protocol to consider fToken worth $1 and when it may be risky. Design and implement mitigations, if necessary, and perform invariant testing on the changes.

17. Rounding direction for deposits does not favor the protocol

Severity: **Undetermined**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ADFX-17
Target: `contracts/f(x)/math/FxStableMath.sol`	

Description

When users mint fToken and xToken, they must transfer the amount of base token obtained by `maxMintableFToken` and `maxMinatableXToken`, respectively, as collateral. Currently, these functions round the value of `_maxBaseIn`, the amount of collateral, down due to integer division. Thus, in some cases, the amount of collateral required may be less than the amount required if the same calculation were done using real numbers, and the protocol will receive less collateral than expected. This requires further investigation, but it would be best to specify rounding directions explicitly even if this issue is not currently exploitable.

```
function maxMintableFToken(SwapState memory state, uint256 _newCollateralRatio)
  internal
  pure
  returns (uint256 _maxBaseIn, uint256 _maxFTokenMintable)
{
[...]
  uint256 _baseVal = state.baseSupply * (state.baseNav) * (PRECISION);
  uint256 _fVal = _newCollateralRatio * (state.fSupply) * (PRECISION);

  if (_baseVal > _fVal) {
    _newCollateralRatio = _newCollateralRatio - (PRECISION);
    uint256 _delta = _baseVal - _fVal;

    _maxBaseIn = _delta / (state.baseNav * (_newCollateralRatio));
    _maxFTokenMintable = _delta / (PRECISION * (_newCollateralRatio));
  }
}
```

Figure 17.1: The calculated base amount required as collateral is rounded down (aladdin-v3-contracts/contracts/f(x)/math/FxStableMath.sol#45–70)

Recommendations

Short term, determine whether the protocol should round up deposits and implement the appropriate behavior, avoiding rounding errors that may trap the system.

Long term, specify rounding directions explicitly and use fuzz testing to identify if it is possible to exacerbate the rounding issue and profit.

18. Reverting when minting xToken can prevent re-collateralization

Severity: **Medium**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-ADFX-18
Target: `contracts/f(x)/v2/TreasuryV2.sol`	

Description

When the treasury processes the `mintXToken` action, it validates that the oracle price is valid, and, if invalid, it reverts. During stability mode (collateral ratio between 100% and 130%), the protocol is designed to recover by incentivizing the minting of xToken in order to re-collateralize above 130%. Minting xToken may fail, even though it is necessary to recover from stability mode, and cause the protocol to collapse. Instead, the protocol could always permit the minting of xToken during stability mode and handle invalid prices by conservatively pricing the collateral and favoring the protocol's credit risk health at the expense of the user minting.

```
if (_action == Action.MintFToken || _action == Action.MintXToken) {
  if (!_isValid) revert ErrorInvalidOraclePrice();
}
```

Figure 18.1: Minting xToken fails if the price is considered invalid (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#648–649)

Recommendations

Short term, consider allowing minting of xToken at a conservative price when the protocol is in stability mode.

Long term, identify actions that should never fail to succeed, such as those relevant to re-collateralizing the protocol, and implement invariant testing that ensures that they always succeed.

19. Unclear economic sustainability of allowing user to avoid liquidations

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ADFX-19
Target: `contracts/f(x)/v2/FxUSD.sol`	

Description

The FxUSD contracts allow moving fToken from the rebalance pool into them and minting FxUSD. (This method is used to attempt to increase the collateral ratio by burning fToken.) Moving tokens in this way is allowed even when liquidations are possible (i.e., the collateral ratio is sufficiently low). The rebalance pool is intended to socialize the losses among fToken depositors, but nothing prevents withdrawing prior to liquidation transactions. This does not ensure that all depositors are on the hook for liquidation.

Users who front-run liquidations and withdraw from the rebalance pool will not have the liquidated collateral credited to their share of rewards from the rebalance pool, but they will retain their balance of fToken. Since redeeming fToken may be the only action for re-collateralizating the protocol if xToken minting is failing (TOB-ADFX-18), allowing withdrawals during stability mode may threaten the ability of the protocol to recover from delinquency.

```
function wrapFrom(
  address _pool,
  uint256 _amount,
  address _receiver
) external override onlySupportedPool(_pool) {
  if (isUnderCollateral()) revert ErrorUnderCollateral();

  address _baseToken = IFxShareableRebalancePool(_pool).baseToken();
  _checkBaseToken(_baseToken);
  _checkMarketMintable(_baseToken, false);

  IFxShareableRebalancePool(_pool).withdrawFrom(_msgSender(), _amount,
address(this));
  _mintShares(_baseToken, _receiver, _amount);

  emit Wrap(_baseToken, _msgSender(), _receiver, _amount);
}
```

Figure 20.1: Users can withdraw from rebalance pool and mint FxUSD in stability mode (aladdin-v3-contracts/contracts/f(x)/v2/FxUSD.sol#160–175)

```
function liquidate(uint256 _maxAmount, uint256 _minBaseOut)
  external
  override
  onlyRole(LIQUIDATOR_ROLE)
  returns (uint256 _liquidated, uint256 _baseOut)
{
[...]

  // distribute liquidated base token
  _accumulateReward(_token, _baseOut);

  // notify loss
  _notifyLoss(_liquidated);
}
```

Figure 20.1: Users who evade liquidation miss rewards and avoid losses (aladdin-v3-contracts/contracts/f(x)/rebalance-pool/ShareableRebalancePoolV2.sol#35–82)

Recommendations

Short term, consider preventing withdrawals from the rebalance pool when the fToken's treasury's collateral ratio is below the stability ratio and thus able to be liquidated.

Long term, perform additional economic analysis regarding liquidations and whether the incentives of fToken holders and FxUSD holders align with the protocol's interests.

20. Validation of system invariants is error prone

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ADFX-20
Target: `contracts/f(x)/v2/TreasuryV2.sol`, `contracts/f(x)/v2/MarketV2.sol`, `contracts/f(x)/v2/FxUSD.sol`	

Description

The architecture of the f(x) protocol is interleaved and its interactions are complex. The interdependencies of the components are not managed well, and validations are distributed in disparate components rather than maintaining logical separation. We recommend simplifying the architecture and refactoring assertions about how the state is updated to be closely tied to where the state is updated. This is especially important in light of the concerns related to stability and incentives and lack of specification, as logical errors and economic issues may be coupled together. Making these investments will strengthen the security posture of the system and facilitate invariant testing.

The Treasury does not validate the postcondition that the collateral ratio increases when minting xToken and redeeming fToken. In stability mode, liquidations perform fToken redemptions to attempt re-collateralizing the protocol to a collateral ratio above the stability ratio.

```
function redeemFToken(
  uint256 _fTokenIn,
  address _recipient,
  uint256 _minBaseOut
) external override nonReentrant returns (uint256 _baseOut, uint256 _bonus) {
  if (redeemPaused()) revert ErrorRedeemPaused();

  if (_fTokenIn == type(uint256).max) {
    _fTokenIn = IERC20Upgradeable(fToken).balanceOf(_msgSender());
  }
  if (_fTokenIn == 0) revert ErrorRedeemZeroAmount();

  uint256 _stabilityRatio = stabilityRatio();
  (uint256 _maxBaseOut, uint256 _maxFTokenInBeforeSystemStabilityMode) =
IFxTreasuryV2(treasury).maxRedeemableFToken(
    _stabilityRatio
  );
  uint256 _feeRatio = _computeFTokenRedeemFeeRatio(_fTokenIn,
_maxFTokenInBeforeSystemStabilityMode);
```

```
  _baseOut = IFxTreasuryV2(treasury).redeem(_fTokenIn, 0, _msgSender());
```

Figure 21.1: Treasury lacks postcondition that collateral ratio increases for fToken redemptions (aladdin-v3-contracts/contracts/f(x)/v2/MarketV2.sol#301–319)

The Treasury does not validate the postcondition that the collateral ratio is not below stability ratio, allowing immediate liquidations when minting fToken. The credit health is softly enforced by the `maxMintableFToken` math library function used in the Market and by the FxUSD contract (when enabled), but it would be more robust to strictly require the collateral ratio has not fallen too far when performing mint actions to prevent insolvency. This applies to redeeming xToken as well since it is also expected to reduce the collateral ratio like minting fToken.

```
function mintFToken(uint256 _baseIn, address _recipient)
  external
  override
  onlyRole(FX_MARKET_ROLE)
  returns (uint256 _fTokenOut)
{
  FxStableMath.SwapState memory _state = _loadSwapState(Action.MintFToken);
  if (_state.xNav == 0) revert ErrorUnderCollateral();
  if (_state.baseSupply + _baseIn > baseTokenCap) revert ErrorExceedTotalCap();

  _updateEMALeverageRatio(_state);

  _fTokenOut = _state.mintFToken(_baseIn);
  totalBaseToken = _state.baseSupply + _baseIn;

  IFxFractionalTokenV2(fToken).mint(_recipient, _fTokenOut);
}
```

Figure 21.2: Treasury lacks postcondition that collateral ratio is greater than stability ratio (aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#286–303)

While the Market charges extra fees when too much is minted, the fees aren't deposited as collateral in the Treasury, and this added complexity increases the likelihood of bugs. In pursuit of simplicity, we'd recommend disallowing minting excess beyond `_maxBaseInBeforeSystemStabilityMode` and only allowing minting up to the maximum amount. Currently, the minted amount is only bound if the `fTokenMintPausedInStabilityMode` is explicitly enabled but not by default. This has the added benefit of making validation and testing/verification easier to perform.

```
(uint256 _maxBaseInBeforeSystemStabilityMode, ) =
IFxTreasuryV2(treasury).maxMintableFToken(_stabilityRatio);
if (_maxBaseInBeforeSystemStabilityMode > 0) {
  _maxBaseInBeforeSystemStabilityMode = IFxTreasuryV2(treasury).getWrapppedValue(
```

```
    _maxBaseInBeforeSystemStabilityMode
  );
}

if (fTokenMintPausedInStabilityMode()) {
  uint256 _collateralRatio = IFxTreasuryV2(treasury).collateralRatio();
  if (_collateralRatio <= _stabilityRatio) revert
ErrorFTokenMintPausedInStabilityMode();

  // bound maximum amount of base token to mint fToken.
  if (_baseIn > _maxBaseInBeforeSystemStabilityMode) {
    _baseIn = _maxBaseInBeforeSystemStabilityMode;
  }
}

uint256 _amountWithoutFee = _deductFTokenMintFee(_baseIn,
_maxBaseInBeforeSystemStabilityMode);
IERC20Upgradeable(baseToken).safeTransferFrom(_msgSender(), treasury,
_amountWithoutFee);

_fTokenMinted = IFxTreasuryV2(treasury).mintFToken(
  IFxTreasuryV2(treasury).getUnderlyingValue(_amountWithoutFee),
  _recipient
);
```

Figure 21.3: Excess minting is permitted for additional fee (aladdin-v3-contracts/contracts/f(x)/v2/MarketV2.sol#226–249)

Rather than having the Treasury validate that it is solvent, the validation is performed in FxUSD (which requires calling the Market and Treasury). This makes the composability of the system fragile as modifying a contract locally can have far reaching consequences that are not apparent in the scope of the diff. Important validations related to core invariants should be clearly documented and be performed as close as possible to the component they are relevant to.

```
function _checkMarketMintable(address _baseToken, bool _checkCollateralRatio)
private view {
  address _treasury = markets[_baseToken].treasury;
  if (_checkCollateralRatio) {
    uint256 _collateralRatio = IFxTreasuryV2(_treasury).collateralRatio();
    uint256 _stabilityRatio =
IFxMarketV2(markets[_baseToken].market).stabilityRatio();
    // not allow to mint when collateral ratio <= stability ratio
    if (_collateralRatio <= _stabilityRatio) revert ErrorMarketInStabilityMode();
  }
  // not allow to mint when price is invalid
  if (!IFxTreasuryV2(_treasury).isBaseTokenPriceValid()) revert
ErrorMarketWithInvalidPrice();
}
```

Figure 21.4: FxUSD performs validation that should be in the core Treasury (aladdin-v3-contracts/contracts/f(x)/v2/FxUSD.sol#391–401)

Rather than validating that the mint cap of fToken is reached in the FractionalToken implementation, it is done in FxUSD. This does not consider when FxUSD has not been added to the Market and the validation should be done in the token implementation instead. In general, performing validations against other contracts' state variables instead of having the contract maintain the invariant is error prone.

```
if (IERC20Upgradeable(_fToken).totalSupply() > _mintCap) revert
ErrorExceedMintCap();
```

Figure 21.5: FxUSD enforces fToken mint cap instead of the token doing it (aladdin-v3-contracts/contracts/f(x)/v2/FxUSD.sol#422)

Recommendations

Short term, validate that the collateral ratio has increased for minting xToken and redeeming fToken and that minting fToken and redeeming XToken do not put the collateral ratio below that stability ratio, allowing for immediate liquidations. While certain configurations of fees may be sufficient to prevent unexpected behavior, it is more robust to simply prohibit actions which have undesired effects.

Long term, perform validations specific to a component locally rather than sporadically performing validations in separate components. Simplify the architecture by refactoring disjointed components and make each component strictly validate itself. Create specifications for each component and their interactions, and perform invariant testing to ensure the specification matches the implementation.

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
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
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
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
Transaction Ordering	The system's resistance to transaction-ordering attacks

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.

C. Code Quality Recommendations

This appendix lists recommendations that are not directly security-relevant but may improve readability, maintainability, or efficiency.

- **Call `_disableInitializers()` in the constructor of every upgradeable contract so that the initialize method does not remain callable on the implementation contract.**
- **Correct the NatSpec in IFxUSD to say that fToken is deposited in the rebalance pool.**

```
/// @notice Deposit fxUSD to rebalance pool.
/// @param pool The address of rebalance pool.
/// @param amount The amount of fxUSD to use.
/// @param receiver The address of rebalance pool share recipient.
function earn(
  address pool,
  uint256 amount,
  address receiver
) external;

/// @notice Mint fxUSD with base token and deposit to rebalance pool.
/// @param pool The address of rebalance pool.
/// @param amountIn The amount of base token to use.
/// @param receiver The address of rebalance pool recipient.
/// @param minOut The minimum amount of rebalance pool shares should receive.
/// @return amountOut The amount of rebalance pool shares received by the receiver.
function mintAndEarn(
  address pool,
  uint256 amountIn,
  address receiver,
  uint256 minOut
) external returns (uint256 amountOut);
```

(aladdin-v3-contracts/contracts/interfaces/f(x)/IFxUSD.sol#129–150)

- **Consider using `FxUSD.wrapFrom` instead of re-implementing the logic in the router.**

```
function fxRebalancePoolWithdraw(address _pool, uint256 _amountIn) external payable
returns (uint256 _amountOut) {
  address _baseToken = IFxShareableRebalancePool(_pool).baseToken();
  address _fToken = IFxShareableRebalancePool(_pool).asset();
  _amountOut = IERC20Upgradeable(_fToken).balanceOf(address(this));
  IFxShareableRebalancePool(_pool).withdrawFrom(msg.sender, _amountIn,
address(this));
  _amountOut = IERC20Upgradeable(_fToken).balanceOf(address(this)) - _amountOut;
  LibGatewayRouter.approve(_fToken, fxUSD, _amountOut);
```

```
  IFxUSD(fxUSD).wrap(_baseToken, _amountOut, msg.sender);
}
```

(aladdin-v3-contracts/contracts/gateways/facets/FxUSDFacet.sol#250–258)

- **Use the `isUnderCollateral` method instead of re-implementing the logic.**

```
bool _isUnderCollateral = false;
for (uint256 i = 0; i < _numMarkets; i++) {
  _baseTokens[i] = supportedTokens.at(i);
  _supplies[i] = markets[_baseTokens[i]].managed;
  address _treasury = markets[_baseTokens[i]].treasury;
  if (IFxTreasuryV2(_treasury).isUnderCollateral()) _isUnderCollateral = true;
}
```

(aladdin-v3-contracts/contracts/f(x)/v2/FxUSD.sol#277–283)

- **Set the fee values to the calculated amount instead of recomputing them each time.**

```
(uint256 _defaultRatio, int256 _deltaRatio) = fTokenRedeemFeeRatio();
uint256 _feeRatio0 = uint256(int256(_defaultRatio) + _deltaRatio);
uint256 _feeRatio1 = _defaultRatio;
```

(aladdin-v3-contracts/contracts/f(x)/v2/MarketV2.sol#665–667)

- **Update the following code to `_baseVal > _fVal`, as xNav will be zero when `baseVal` is equal to `_fval`.**

```
if (_baseVal >= _fVal) {
  _state.xNav = (_baseVal - _fVal) / _state.xSupply;
} else {
  // under collateral
  _state.xNav = 0;
}
```

(aladdin-v3-contracts/contracts/f(x)/v2/TreasuryV2.sol#622–627)

- **Get token addresses from the market where possible.**

```
constructor(
  address _market,
  address _baseToken,
  address _fToken,
  address _xToken
) {
  baseToken = _baseToken;
  fToken = _fToken;
  xToken = _xToken;
```

```
  market = _market;
}
```

(*aladdin-v3-contracts/contracts/gateways/facets/FxMarketV1Facet.sol#35–45*)

D. 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.

From April 1 to April 3, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Aladdin 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 20 issues described in this report, Aladdin has resolved four issues, has partially resolved one issue, and has not resolved the remaining 15 issues.

ID	Title	Status
1	Redeeming xToken and fToken simultaneously uses incorrect TWAP price	Unresolved
2	liquidatableCollateralRatio update can force liquidation without warning	Resolved
3	Panicking checked arithmetic may prevent deposits and withdraws to rebalance pool	Unresolved
4	Incorrect TWAP price may be used for calculations such as collateral ratio	Unresolved
5	Updating strategy may cause users to lose funds during redemption	Unresolved
6	Time-weighted Chainlink oracle can report inaccurate price	Unresolved
7	Net asset value of fractional and leverage token may reflect invalid price	Partially Resolved
8	Collateral ratio does not account underlying value of collateral in strategy	Unresolved
9	Rewards are withdrawn even if protocol is not sufficiently collateralized	Resolved
10	Rebalance pool withdrawal silently fails	Resolved

11	Upgradeable contract initialization calls are commented out	Unresolved
12	Sum of all user shares does not equal total supply	Unresolved
13	Lack of validation when updating system configurations	Resolved
14	Lack of slippage checks prevents user from specifying acceptable loss	Unresolved
15	Deployments to L2 should check sequencer uptime for Chainlink price feeds	Unresolved
16	Treating fToken as $1 creates arbitrage opportunity and unclear incentives	Unresolved
17	Rounding direction for deposits does not favor the protocol	Unresolved
18	Reverting when minting xToken can prevent re-collateralization	Unresolved
19	Unclear economic sustainability of allowing user to avoid liquidations	Unresolved
20	Validation of system invariants is error prone	Unresolved

Detailed Fix Review Results

TOB-ADFX-1: Redeeming xToken and fToken simultaneously uses incorrect TWAP price
Unresolved.

The client provided the following context for this finding's fix status:

> *In our current setting, the code is unreachable. So it is ok to not fix the issue.*

TOB-ADFX-2: liquidatableCollateralRatio update can force liquidation without warning
Resolved in PR#178. `liquidatableCollateralRatio` was deprecated and replaced with the stability ratio, which is enforced to be at least 100% as of PR#181. This ensures that liquidations can occur only when the protocol's collateralization rate falls below the stability ratio.

The client provided the following context for this finding's fix status:

> *We use `market.stabilityRatio` to avoid inconsistency. As the problem mentioned, we will add a timelock when we switch our governance method.*

TOB-ADFX-3: Panicking checked arithmetic may prevent deposits and withdraws to rebalance pool
Unresolved.

The client provided the following context for this finding's fix status:

> *If the code is correct, `_balance.amount` is always `>= _newBalance`. So in normal case, it won't panic. If it indeed panicked, we will know it has some bug in our code.*

TOB-ADFX-4: Incorrect TWAP price may be used for calculations such as collateral ratio
Unresolved in #PR183. The proposed fix does not calculate the collateral ratio uniformly across the protocol, and the collateral ratio will vary based on the action being performed. This introduces additional complexity and may delay liquidation since the collateral ratio calculated when redeeming fToken uses the maximum price of the collateral, taking the upper limit compared to if the "safe" price is used. We recommend decoupling the calculation of the collateral ratio and the NAV for mint/redeem actions instead of sharing the implementation of `_fetchTwapPrice` and using a uniform collateral throughout the protocol.

TOB-ADFX-5: Updating strategy may cause users to lose funds during redemption
Unresolved.

The client provided the following context for this finding's fix status:

> *This code is not used in our current setting. We will be careful about this if we decide to use the code.*

TOB-ADFX-6: Time-weighted Chainlink oracle can report inaccurate price
Unresolved.

The client provided the following context for this finding's fix status:

> *Due to potential sandwich attack on spot price, we think using twap is a better option here.*

TOB-ADFX-7: Net asset value of fractional and leverage token may reflect invalid price
Partially resolved in PR#179. The API documents that the price must be validated but does not revert or provide a Boolean value indicating the price's validity.

TOB-ADFX-8: Collateral ratio does not account underlying value of collateral in strategy
Unresolved.

The client provided the following context for this finding's fix status:

> *This code is not used in our current setting. We will be careful about this if we decide to use the code.*

TOB-ADFX-9: Rewards are withdrawn even if protocol is not sufficiently collateralized
Resolved in PR#182. Harvesting rewards is disabled when the protocol is undercollateralized.

The client provided the following context for this finding's fix status:

> *In stability mode, harvest won't change collateral ratio. So we decide only disable harvest when under collateral.*

TOB-ADFX-10: Rebalance pool withdrawal silently fails
Resolved in #PR180. The withdrawal fails with a custom error instead of failing silently.

TOB-ADFX-11: Upgradeable contract initialization calls are commented out
Unresolved.

The client provided the following context for this finding's fix status:

> *The commented out codes is empty now. We will be careful about this when upgrading dependency.*

TOB-ADFX-12: Sum of all user shares does not equal total supply
Unresolved.

The client provided the following context for this finding's fix status:

> *The `totalSupply` won't change after initialized. It is ok to keep it current way.*

TOB-ADFX-13: Lack of validation when updating system configurations
Resolved in PR#181. The function that updates Boolean configurations will revert if the update does not have any effect. The stability ratio is validated to be at least 100%.

TOB-ADFX-14: Lack of slippage checks prevents user from specifying acceptable loss
Unresolved.

The client provided the following context for this finding's fix status:

> *We did have slippage checks in the following line.*

TOB-ADFX-15: Deployments to L2 should check sequencer uptime for Chainlink price feeds
Unresolved.

The client provided the following context for this finding's fix status:

> *We will fix this in our new oracle design.*

TOB-ADFX-16: Treating fToken as $1 creates arbitrage opportunity and unclear incentives
Unresolved.

The client provided the following context for this finding's fix status:

> *This kind of arbitrage opportunity is very normal market behavior. We don't treat it as big threat.*

TOB-ADFX-17: Rounding direction for deposits does not favor the protocol
Unresolved.

The client provided the following context for this finding's fix status:

> *We don't see any big threat for now. Will revisit this issue later.*

TOB-ADFX-18: Reverting when minting xToken can prevent re-collateralization
Unresolved.

The client provided the following context for this finding's fix status:

We allow to mint xToken when price is valid during stability mode. When the price is invalid, preventing mint will protect users.

TOB-ADFX-19: Unclear economic sustainability of allowing user to avoid liquidations
Unresolved.

The client provided the following context for this finding's fix status:

It is by design to let users deposit and withdraw without any limitation.

TOB-ADFX-20: Validation of system invariants is error prone
Unresolved.

The client provided the following context for this finding's fix status:

1. Treasury lacks postcondition that collateral ratio increases for fToken redemptions

The increase of collateral ratio is guaranteed by the invariant formula.

2. Treasury lacks postcondition that collateral ratio is greater than stability ratio

The Treasury is not aware of what is stability ratio and it allows minting unless under collateralized. The stability ratio is controlled by Market.

3. Excess minting is permitted for additional fee

That's by design that we implement this in Market/Treasury contract not fxUSD. They are allowed to mint/redeem any time unless under collateral. The additional fee is used to prevent unnecessary mint/redeem. We will revisit the design later.

4. FxUSD performs validation that should be in the core Treasury

FxUSD is an external module for Treasury/Market. The Treasury/Market themselves don't do such check. That's why it is checked in FxUSD.

5. FxUSD enforces fToken mint cap.

FxUSD is an external module.

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.