Security Audit Report

f(x) Protocol Update by AladdinDAO

SECBIT

September 15, 2023

1. Introduction

The AladdinDAO is a decentralized network to shift crypto investments from venture capitalists to the wisdom of crowds through collective value discovery. As a part of the AladdinDAO ecosystem, the f(x) protocol creates two new ETH derivative assets, one with stablecoin-like low volatility called fractional ETH (fETH) and the second a leveraged long ETH perpetual token called leveraged ETH (xETH). Compared to the old version of the f(x) protocol, the current new codes have a batch of changes, including adding additional incentives for xETH minting, implementing new Oracle logic, and supporting L2 blockchains. SECBIT Labs conducted an audit from September 1 to September 15, 2023, including an analysis of the smart contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that this f(x) contract update has no critical security risks. The SECBIT team has some tips on logical implementation, potential risks, and code revising (see part 4 for details).

Type	Description	Level	Status
Code Practice	4.3.1 Add an `emit` event within the `harvest()` function to record the amount of rewards distributed.	Info	Fixed
Gas optimization	4.3.2 Skip the stETH token incentive logic when the protocol's collateralization ratio is above the stability threshold to reduce user gas consumption.	Info	Fixed
Design & Implementation	4.3.3 Discuss the additional stETH token incentive logic when users mint xETH tokens.	Info	Discussed
Design & Implementation	4.3.4 Discussion on the Meaning of the `baseSwapAmt` Parameter.	Info	Fixed
Code Revise	4.3.5 The function name should indeed be corrected from `_defuctMintFee` to `_deductMintFee`.	Info	Fixed
Design & Implementation	4.3.6 The `liquidate()` function may potentially fail.	Low	Discussed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about the f(x) Protocol update is shown below:

- Smart contract code
  - initial review commit *2bbc9d4*
  - final review commit *40f56e7*

This audit focuses on the new features added to the protocol, with specific code changes starting from commit: *05bd9cc*.

2.2 Contract List

The following content shows the contracts included in the f(x) Protocol update, which the SECBIT team audits:

Name	Lines	Description
FxETHTwapOracle.sol	74	Provide base token quotations for the protocol.
ChainlinkWstETHRateProvider.sol	12	Retrieve the price of the wstETH-ETH trading pair under the Chainlink protocol.
wBETHProvider.sol	12	Retrieve the price of the wBETH-ETH trading pair.
HarvestableTreasury.sol	66	Harvest pending rewards to stability pool.
Market.sol	502	A contract that implements the core logic of the f(x) protocol.
ReservePool.sol	99	A contract has been established to store a portion of the protocol's earnings, and administrators can perform liquidation using the base token held within the contract.
Treasury.sol	392	A contract to store the baseToken, where the core functions can only be called by the Market contract.
WrappedTokenTreasury.sol	11	Harvest pending rewards to stability pool.
FxGateway.sol	167	Extend the use cases of the fx protocol to allow users to participate in the protocol using approved tokens.
MultiPathConverter.sol	39	A contract for token exchange.

Notice: This audit specifically focuses on the new code introduced in the `Market` *and* `Treasury` *contracts.*

3. Contract Analysis

This part describes code assessment details, including "role classification" and "functional analysis".

3.1 Role Classification

Three key roles in the f(x) Protocol are Governance Account and Common Account.

- Governance Account
    - Description

      Contract Administrator
    - Authority
        - Update fees ratio
        - Update market and incentive configurations
        - Transfer ownership
        - Pause crucial functions
    - Method of Authorization

      The contract administrator is the contract's creator or authorized by transferring the governance account.
- Common Account
    - Description

      Users participate in the f(x) Protocol.
    - Authority
        - Mint fETH / xETH with authorized tokens

        - Redeem base tokens with fETH / xETH tokens
        - Add base tokens to increase the collateralization ratio
        - Liquidate fETH tokens to increase the collateralization ratio
        - Swap between fETH and xETH
    - Method of Authorization

      No authorization required

3.2 Functional Analysis

The f(x) protocol implements a decentralized quasi-stablecoin with high collateral utilization efficiency and leveraged contracts with low liquidation risks and no funding costs. The new version of the contract has expanded the protocol's use cases and introduced new features. The SECBIT team conducted a detailed audit of some of the contracts in the protocol. We can divide the critical functions of the contract into three parts:

ReservePool

As a reserve pool of funds, this contract stores a portion of the protocol's fee income. When the protocol's collateralization ratio falls below the stability threshold, the keeper triggers an incentive mechanism, allowing users to receive additional rewards when minting xTokens. The funds in this contract will also be used to repurchase xTokens from the market during liquidation.

The main functions in this contract are as below:

- `requestBonus()`

  When the collateralization ratio falls below the stability threshold, the keeper will adjust the relevant parameters and reward users who mint xTokens.
- `liquidate()`

  The keeper can use the reserve funds in this contract to perform liquidation to increase the protocol's collateralization ratio.

FXGateway

As a peripheral auxiliary contract, this contract provides users a more user-friendly interface. The main functions in this contract are as below:

- `mintFToken()`

  Anyone could call this function to mint some fTokens with some authorized tokens under certain conditions.

- `mintXToken()`

  Anyone could call this function to mint some xTokens with some authorized tokens under certain conditions.

- `addBaseToken()`

  This function allows users to add baseTokens, minting xTokens in the process. At the same time, it adjusts the system's Net Asset Value (NAV) to increase the collateralization ratio.

- `redeem()`

  Users can redeem fTokens and xTokens and receive baseTokens through this function.

- `swap()`

  Users can swap between fTokens and xTokens.

Market, StableCoinMath, and Treasury

These contracts implement the core functionality of the f(x) protocol. The main functions in these contracts are as below:

- `mint()`

  This function can only be called once during the protocol's initialization to ensure the contract's regular operation.

- `mintFToken()`

  This function is used to mint fTokens, which may decrease the system's collateralization ratio.

- `mintXToken()`

  This function is used to mint xTokens, which increases the system's collateralization ratio and contributes to the stability of the system.

- `addBaseToken()`

  Users can call this function to add more baseTokens when the collateralization ratio falls below the specific threshold, aiming to maintain system stability. Additionally, users can earn rewards for their participation.

- `redeem()`

  By using this function, users can burn fTokens and xTokens, and in return, they will receive the corresponding baseTokens.

- `liquidate()`

  When the system's collateralization ratio falls below the liquidation threshold, users can utilize this function to perform asset liquidation, and they will also receive a certain percentage of rewards as incentives.

- `selfLiquidate()`

  When the system's collateralization ratio further decreases, the protocol administrator can call this function to perform asset liquidation, preventing the protocol from entering recapping mode.

4. Audit Detail

This part describes the process, and the detailed audit results also demonstrate the problems and potential risks.

4.1 Audit Process

The audit strictly followed the audit specification of SECBIT Lab. We analyzed the project from code bugs, logical implementation, and potential risks. The process consists of four steps:

- Fully analysis of contract code line by line.
- Evaluation of vulnerabilities and potential risks revealed in the contract code.
- Communication on assessment and confirmation.
- Audit report writing.

4.2 Audit Result

After scanning with adelaide, sf-checker, and badmsg.sender (internal version) developed by SECBIT Labs and open source tools, including Mythril, Slither, SmartCheck, and Securify, the auditing team performed a manual assessment. The team inspected the contract line by line, and the result could be categorized into the following types:

Number	Classification	Result
1	Normal functioning of features defined by the contract	✓
2	No obvious bug (e.g., overflow, underflow)	✓
3	Pass Solidity compiler check with no potential error	✓
4	Pass common tools check with no obvious vulnerability	✓
5	No obvious gas-consuming operation	✓
6	Meet with ERC20 standard	✓

7	No risk in low-level call (call, delegatecall, callcode) and in-line assembly	✓
8	No deprecated or outdated usage	✓
9	Explicit implementation, visibility, variable type, and Solidity version number	✓
10	No redundant code	✓
11	No potential risk manipulated by timestamp and network environment	✓
12	Explicit business logic	✓
13	Implementation consistent with annotation and other info	✓
14	No hidden code about any logic that is not mentioned in the design	✓
15	No ambiguous logic	✓
16	No risk threatening the developing team	✓
17	No risk threatening exchanges, wallets, and DApps	✓
18	No risk threatening token holders	✓
19	No privilege on managing others' balances	✓
20	No non-essential minting method	✓
21	Correct managing hierarchy	✓

4.3 Issues

4.3.1 Add an `emit` event within the `harvest()` function to record the amount of rewards distributed.

Risk Type	Risk Level	Impact	Status
Code Practice	Info	Design logic	Fixed

Description

In Ethereum, events are essential components of a contract, primarily used for recording various runtime activities. When an event is emitted, it stores the arguments passed within transaction logs, making these logs accessible to external analytics and reporting tools. The `HarvestableTreasury` contract has declared the corresponding `Harvest` event but has not used it in the `harvest()` function. To ensure proper logging of the harvesting event, it is advisable to include an "emit Harvest" statement within the `harvest()` function.

```
// @audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/HarvestableTreasury.sol#L79-L104

/// @notice Emitted when someone harvest pending stETH
rewards.
/// @param caller The address of caller.
/// @param totalRewards The amount of total harvested rewards.
/// @param stabilityPoolRewards The amount of harvested
rewards distributed to stability pool.
/// @param harvestBounty The amount of harvested rewards
distributed to caller as harvest bounty.
event Harvest(address indexed caller, uint256 totalRewards,
uint256 stabilityPoolRewards, uint256 harvestBounty);
```

```
/// @notice Harvest pending rewards to stability pool.
function harvest() external {
  address _baseToken = baseToken;

  uint256 _totalRewards =
IERC20Upgradeable(_baseToken).balanceOf(address(this)).sub(
    convertToWrapped(totalBaseToken)
  );
  uint256 _harvestBounty = (harvestBountyRatio *
_totalRewards) / PRECISION;
  uint256 _stabilityPoolRewards = (stabilityPoolRatio *
_totalRewards) / PRECISION;

  ......
}
```

Suggestion

Add emit event in the `harvest()` function:

```
function harvest() external {
  address _baseToken = baseToken;

  uint256 _totalRewards =
IERC20Upgradeable(_baseToken).balanceOf(address(this)).sub(
    convertToWrapped(totalBaseToken)
  );
  uint256 _harvestBounty = (harvestBountyRatio *
_totalRewards) / PRECISION;
  uint256 _stabilityPoolRewards = (stabilityPoolRatio *
_totalRewards) / PRECISION;

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

  //@audit add the event
```

```
    emit Harvest(msg.sender, _totalRewards,
_stabilityPoolRewards, _harvestBounty);
}
```

Status

The development team has adopted the suggestion and fixed this issue in commit *154277b*.

4.3.2 Skip the stETH token incentive logic when the protocol's collateralization ratio is above the stability threshold to reduce user gas consumption.

Risk Type	Risk Level	Impact	Status
Gas optimization	Info	More gas consumption	Fixed

Description

According to the Aladdin f(x) requirements document: "When a user mints xETH, they receive a certain percentage of stETH incentives. The rewards are payed from the reserve pool." "Assuming a user mints xETH using n stETH, the user can receive a maximum of n*5% in stETH incentives. Any excess stETH beyond this threshold will not be eligible for rewards when the system is recovered. In essence, users are motivated to mint xETH to keep the CR>=130%."

In the current code design, the parameter `_amountWithoutFee` represents the amount of stETH that the user has deposited (after deducting the fees), and the parameter `_maxBaseInBeforeSystemStabilityMode` means the amount of stETH that the user needs to deposit to bring the protocol back to stability. If the protocol is already in a stable state, then the value of this parameter is zero. Suppose the protocol is already in a stable state, meaning `_maxBaseInBeforeSystemStabilityMode` is zero. In this case, users do not receive additional incentives, and there is no need to execute the

`requestBonus()` function, which can reduce user gas consumption.

```
// @audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/Market.sol#L300-L335

function mintXToken(
    uint256 _baseIn,
    address _recipient,
    uint256 _minXTokenMinted
  ) external override nonReentrant returns (uint256
_xTokenMinted) {
    ......
    require(_xTokenMinted >= _minXTokenMinted, "insufficient
xToken output");

    // give bnous
    // @audit provide additional stETH incentives to users
when the system's collateralization ratio falls below the
stability threshold.
    if (_amountWithoutFee <
_maxBaseInBeforeSystemStabilityMode) {
      IReservePool(reservePool).requestBonus(baseToken,
_recipient, _amountWithoutFee);
    } else {
      IReservePool(reservePool).requestBonus(baseToken,
_recipient, _maxBaseInBeforeSystemStabilityMode);
    }

    emit Mint(msg.sender, _recipient, _baseIn, 0,
_xTokenMinted, _baseIn - _amountWithoutFee);
  }
```

Status

The development team has adopted the suggestion and fixed this issue in commit *154277b*.

4.3.3 Discuss the additional stETH token incentive logic when users mint xETH tokens.

Risk Type	Risk Level	Impact	Status
Design & Implementation	Info	Design logic	Discussed

Description

It needs to be confirmed whether the description in the design requirements document aligns with the code implementation. It shows that when the CR falls below 130% (the warning line), it enters a risk control mode, and keepers can trigger the liquidation of the Rebalance pool. If all fETH tokens in the Rebalance pool are liquidated and the CR remains below 130%, the system incentivizes users to mint xETH tokens to increase the system's CR to at least 130%. However, in the current code, users receive stETH tokens rewards whenever the protocol's collateralization ratio is below the stability ratio, not necessarily after a keeper-triggered Rebalance pool liquidation. Therefore, it's essential to verify whether the code aligns with the described design intent in the document.

```
// @audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/Market.sol#L300-L335

function mintXToken(
    uint256 _baseIn,
    address _recipient,
    uint256 _minXTokenMinted
  ) external override nonReentrant returns (uint256
_xTokenMinted) {
    ......
    require(_xTokenMinted >= _minXTokenMinted, "insufficient
xToken output");

    // give bnous
```

```
    // @audit provide additional stETH incentives to users
when the system's collateralization ratio falls below the
stability threshold
    if (_amountWithoutFee <
_maxBaseInBeforeSystemStabilityMode) {
      IReservePool(reservePool).requestBonus(baseToken,
_recipient, _amountWithoutFee);
    } else {
      IReservePool(reservePool).requestBonus(baseToken,
_recipient, _maxBaseInBeforeSystemStabilityMode);
    }

    emit Mint(msg.sender, _recipient, _baseIn, 0,
_xTokenMinted, _baseIn - _amountWithoutFee);
  }
```

Status

According to the code logic, when the Rebalance pool has not been completely liquidated, setting the `bonusRatio` parameter to 0 will indeed prevent users from receiving additional rewards when minting xTokens. An administrator can manually adjust this to align with the logic described in the document.

4.3.4 Discussion on the Meaning of the `baseSwapAmt` Parameter.

Risk Type	Risk Level	Impact	Status
Design & Implementation	Info	Design logic	Fixed

Description

Based on the explanation provided in the `selfLiquidate()` function, where it states, "The amount of base token to swap," it indicates that the parameter `baseSwapAmt` represents the quantity of base tokens used for liquidation. However, in the current version, due to the added support for other chains, the type of base token deployed on L2 differs from the ETH mainnet. On the

mainnet, the base token is stETH, while on Arbitrum and OP, it is wstETH. Additionally, many parameters involve the conversion between wrapped and unwrapped tokens. Therefore, in this function, this parameter may not necessarily represent the actual quantity of the base token. When deployed on L2, this parameter defines the amount after conversion through the `convertToUnwrapped()` function (stETH) and may not be the quantity of the L2 base token. Hence, it is advisable to modify the comments for the corresponding parameter `baseSwapAmt` to reflect the actual scenario accurately.

```
// @audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/interfaces/IMarket.sol#L139
/// @notice Self liquidate some fToken to increase the
collateral ratio.
/// @param baseSwapAmt The amount of base token to swap.
/// @param minFTokenLiquidated The minimum amount of fToken
should be liquidated.
/// @param data The data used to swap base token to fToken.
/// @return baseOut The amount of base token should be
received.
/// @return fTokenLiquidated the amount of fToken liquidated.
function selfLiquidate(
  uint256 baseSwapAmt,
  uint256 minFTokenLiquidated,
  bytes calldata data
) external returns (uint256 baseOut, uint256
fTokenLiquidated);


//@audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/Market.sol#L500
function selfLiquidate(
    uint256 _baseSwapAmt,
    uint256 _minFTokenLiquidated,
    bytes calldata _data
```

```
  ) external override nonReentrant returns (uint256 _baseOut,
uint256 _fTokenLiquidated) {
    require(!redeemPaused, "redeem is paused");
    require(liquidationWhitelist[msg.sender], "not liquidation
whitelist");

    ITreasury _treasury = ITreasury(treasury);
    uint256 _collateralRatio = _treasury.collateralRatio();

    MarketConfig memory _marketConfig = marketConfig;
    require(
      _marketConfig.recapRatio <= _collateralRatio &&
_collateralRatio < _marketConfig.selfLiquidationRatio,
      "Not self liquidation mode"
    );

    // bound the amount of base token
    (uint256 _maxBaseOut, ) = _treasury.maxLiquidatable(
      _marketConfig.selfLiquidationRatio,
      incentiveConfig.selfLiquidationIncentiveRatio
    );
    if (_baseSwapAmt > _maxBaseOut) {
      _baseSwapAmt = _maxBaseOut;
    }

    ......
  }
```

Status

The development team has adopted the suggestion and fixed this issue in commit *154277b*.

4.3.5 The function name should indeed be corrected from `_defuctMintFee` to `_deductMintFee`.

Risk Type	Risk Level	Impact	Status
Code Revise	Info	Wrong Spell	Fixed

Description

See the title.

```
// @audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/Market.sol#L732
_baseInWithoutFee = _defuctMintFee(_baseIn, _feeRatio0,
_feeRatio1, _maxBaseInBeforeSystemStabilityMode);

// @audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/Market.sol#L751
_baseInWithoutFee = _defuctMintFee(_baseIn, _feeRatio0,
_feeRatio1, _maxBaseInBeforeSystemStabilityMode);


//@audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/Market.sol#L754
function _defuctMintFee(
    uint256 _baseIn,
    uint256 _feeRatio0,
    uint256 _feeRatio1,
    uint256 _maxBaseInBeforeSystemStabilityMode
  ) internal returns (uint256 _baseInWithoutFee) {
    uint256 _maxBaseIn =
_maxBaseInBeforeSystemStabilityMode.mul(PRECISION).div(PRECISI
ON - _feeRatio0);
```

```
    // compute fee
    uint256 _fee;
    if (_baseIn <= _maxBaseIn) {
      _fee = _baseIn.mul(_feeRatio0).div(PRECISION);
    } else {
      _fee = _maxBaseIn.mul(_feeRatio0).div(PRECISION);
      _fee = _fee.add((_baseIn -
_maxBaseIn).mul(_feeRatio1).div(PRECISION));
    }

    _baseInWithoutFee = _baseIn.sub(_fee);
    // take fee to platform
    if (_fee > 0) {

 IERC20Upgradeable(baseToken).safeTransferFrom(msg.sender,
platform, _fee);
    }
  }
```

Status

The development team has fixed this issue in commit *154277b*.

4.3.6 The `liquidate()` function may potentially fail.

Risk Type	Risk Level	Impact	Status
Design & Implementation	Low	Design logic	Discussed

Description

When the protocol's collateralization ratio falls below the liquidation threshold, anyone can call the Market.liquidate() function to perform liquidation and maintain the system's security. Users could only use the exact amount of fTokens required to bring the protocol up to the liquidation threshold. You can see this logic in the code here: GitHub Code.

If the ReservePool contract holds more fTokens than required to reach the liquidation threshold, any unused fTokens will remain in it. In this case, the required condition will not be met, and the liquidation action will fail for safety reasons.

```
// @audit link: https://github.com/AladdinDAO/aladdin-v3-
contracts/blob/58d6436c325e9529f11df7faa8ca59ebbbd00809/contra
cts/f(x)/ReservePool.sol#L128

function liquidate(ZapInCall memory _call, uint256
_minBaseOut) external returns (uint256 _baseOut) {
    require(hasRole(LIQUIDATOR_ROLE, msg.sender), "only
liquidator");

    bool _success;
    if (_call.src == address(0)) {
      (_success, ) = _call.target.call{ value: _call.amount }
(_call.data);
    } else {
      IERC20(_call.src).safeApprove(_call.target, 0);
      IERC20(_call.src).safeApprove(_call.target,
_call.amount);
      (_success, ) = _call.target.call(_call.data);
    }

    // below lines will propagate inner error up
    if (!_success) {
      // solhint-disable-next-line no-inline-assembly
      assembly {
        let ptr := mload(0x40)
        let size := returndatasize()
        returndatacopy(ptr, 0, size)
        revert(ptr, size)
      }
    }

    uint256 _fTokenIn =
IERC20(fToken).balanceOf(address(this));
    IERC20(fToken).safeApprove(market, 0);
```

```
    IERC20(fToken).safeApprove(market, _fTokenIn);
    _baseOut = IMarket(market).liquidate(_fTokenIn,
address(this), _minBaseOut);

    // make sure all fToken is used to prevent liquidator
steal fund.
    // @audit the fToken in the contract might not be fully
utilized, which can directly result in revert
    require(IERC20(fToken).balanceOf(address(this)) == 0, "has
dust fToken");
  }
```

Status

This function was removed in commit 40f56e7 according to the discussion result.

5. Conclusion

After auditing and analyzing the f(x) Protocol update, SECBIT Labs found some issues to optimize and proposed corresponding suggestions, which have been shown above.

Disclaimer

SECBIT smart contract audit service assesses the contract's correctness, security, and performability in code quality, logic design, and potential risks. The report is provided "as is", without any warranties about the code practicability, business model, management system's applicability, and anything related to the contract adaptation. This audit report is not to be taken as an endorsement of the platform, team, company, or investment.

APPENDIX

Vulnerability/Risk Level Classification

Level	Description
High	Severely damage the contract's integrity and allow attackers to steal Ethers and tokens, or lock assets inside the contract.
Medium	Damage contract's security under given conditions and cause impairment of benefit for stakeholders.
Low	Cause no actual impairment to contract.
Info	Relevant to practice or rationality of the smart contract, could possibly bring risks.

SECBIT Lab is devoted to constructing a common-consensus, reliable, and ordered blockchain economic entity.

https://secbit.io

audit@secbit.io

@secbit_io