STRONGHOLD
SECURITY

Nether.Fi Security Audit Report

November 29, 2023

Contents

Executive Summary

Title	Description
Client	Nether.Fi
Project	Nether.Fi Contracts
Platform	Base Network
Language	Solidity
Repository	nether.fi-contracts
Initial commit	e2531589be47e5530cc2157e6eb2af9119a03c2b
Final commit	-
Timeline	October 02 2023 - November 13 2023

Project Overview

Nether.Fi is a decentralized perpetual & swap exchange on Base Network.

Audit Scope

File	Link
Vault.sol	Vault.sol
VaultPriceFeed.sol	VaultPriceFeed.sol
NlpManager.sol	NlpManager.sol
USDG.sol	USDG.sol
EsNEFI.sol	EsNEFI.sol
YieldToken.sol	YieldToken.sol
NEFI.sol	NEFI.sol
NLP.sol	NLP.sol
BaseToken.sol	BaseToken.sol
MintableBaseToken.sol	MintableBaseToken.sol
BasePositionManager.sol	BasePositionManager.sol
PositionManager.sol	PositionManager.sol
PositionRouter.sol	PositionRouter.sol

Audit Methodology

General Code Assessment

The code is reviewed for clarity, consistency, style, and whether it follows code best practices applicable to the particular programming language used, such as indentation, naming convention, commented code blocks, code duplication, confusing names, irrelevant
or missing comments, etc. This part is aimed at understanding the overall code structure and protocol architecture. Also, it seeks to learn overall system architecture and business logic and how different parts of the code are related to each other.

Code Logic Analysis

The code logic of particular functions is analyzed for correctness and efficiency. The code is checked for what it is intended for, the algorithms are optimal and valid, and the correct data types are used. The external libraries are checked for relevance and correspond to the tasks they solve in the code. This part is needed to understand the data structures used and the purposes for which they are used. At this stage, various public checklists are applied in order to ensure that logical flaws are detected.

Entities and Dependencies Usage Analysis

The usages of various entities defined in the code are analyzed. This includes both: internal usage from other parts of the code as well as possible dependencies and integration usage. This part aims to understand and spot overall system architecture flaws and bugs in integrations with other protocols.

Access Control Analysis

Access control measures are analyzed for those entities that can be accessed from outside. This part focuses on understanding user roles and permissions, as well as which assets should be protected and how.

Use of checklists and auditor tools

Auditors can perform a more thorough check by using multiple public checklists to look at the code from different angles. Static analysis tools (Slither) help identify simple errors and highlight potentially hazardous areas. While using Echidna for fuzz testing will speed up the testing of many invariants, if necessary.

Vulnerabilities

The audit is directed at identifying possible vulnerabilities in the project's code. The result of the audit is a report with a list of detected vulnerabilities ranked by severity level:

Severity	Description
Critical	Vulnerabilities leading to the theft of assets, blocking access to funds, or any other loss of funds.
High	Vulnerabilities that cause the contract to fail and that can only be fixed b y modifying or completely replacing the contract code.
Medium	Vulnerabilities breaking the intended contract logic but without loss of fun ds and need for contract replacement.
Low	Minor bugs that can be taken into account in order to improve the overall qu ality of the code

After the stage of bug fixing by the Customer, the findings can be assigned t he following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect it s security.
Acknowledged	The Customer took into account the finding. However, the recommendations wer e not implemented since they did not affect the project's safety.

Findings Summary

Severity	# of Findings
Critical	0
High	3
Medium	12
Low	16

ID	Severity	Title	Status
H-1	High	The getAum function can include the same token twice	Acknowledged
H-2	High	Incorrect usage of the Chainlink feed	Acknowledged
H-3	High	The protocol disadvantage in case of stable token depeg	Acknowledged
M-1	Medium	A possibility of DoS (`allWhitelistedTokens`)	Acknowledged
M-2	Medium	Tokens cannot be removed from the whitelist	Acknowledged
M-3	Medium	Users may not pay a deposit fee	Acknowledged
M-4	Medium	A possibility to remove liquidity without the cooldown duration	Acknowledged
M-5	Medium	There are no limits for variables	Acknowledged
M-6	Medium	Positions and token balances are lost in the `Vault` upgrade	Acknowledged
M-7	Medium	A fixed fee reward	Acknowledged
M-8	Medium	A possibility to lose access for governance	Acknowledged
M-9	Medium	Access to user token transfers	Acknowledged
M-10	Medium	A problem with the access control	Acknowledged
M-11	Medium	Incorrect liquidity adding when transferring a few tokens	Acknowledged

M-12	Medium	USDG tokens can be bought and sold in one transaction	Acknowledged
L-1	Low	Naming duplication	Acknowledged
L-2	Low	Old Solidity version and floating pragma	Acknowledged
L-3	Low	Events for the setter functions	Acknowledged
L-4	Low	Stucked ETH transfer out	Acknowledged
L-5	Low	Unsafe default price feed	Acknowledged
L-6	Low	Input value checks	Acknowledged
L-7	Low	Chainlink flags	Acknowledged
L-8	Low	The constructor creates two roles for one account	Acknowledged
L-9	Low	No NatSpec	Acknowledged
L-10	Low	Out of gas in possible infinite loops	Acknowledged
L-11	Low	No array length check	Acknowledged
L-12	Low	The allowance double-spend exploit	Acknowledged
L-13	Low	The name and symbol of the token can be changed	Acknowledged
L-14	Low	Possible overflow	Acknowledged
L-15	Low	Division by zero	Acknowledged
L-16	Low	A Base network compatibility issue	Acknowledged

Findings

Critical

Not Found

High

H-1 High The getAum function can include the same token twice Acknowledged

Description

NlpManager.sol#L137-L144

Vault.sol#L360-L405

The `getAum` function in the `NlpManager` contract iterates all tokens from the `vault` in the `allWhitelistedTokens` array.

In `vault` two functions exist: `setTokenConfig` and `clearTokenConfig`.

The first function registers the token, and the second function deletes the registered token.
The `setTokenConfig` function adds a token to the `allWhitelistedTokens` array.

So, let's consider the following case:

1. The admin registers a token using `setTokenConfig` – `allWhitelistedTokens` have one token;
2. The admin deletes the token using `clearTokenConficg` – here, the token from `allWhitelistedTokens` isn't removed; therefore, `allWhitelistedTokens` will have one token;
3. The admin adds the same token using `setTokenConfig` – the token will be added to `allWhitelistedTokens` again, and `allWhitelistedTokens` will have **TWO** tokens;
4. When someone calls `NlpManager#getAum`, this line `uint256 length = vault.allWhitelistedTokensLength();` – will return an incorrect result 2 instead of 1;
5. Users will run the code in for-loop twice, but it must be run only once.

Recommendation

We recommend removing the token from `allWhitelistedTokens` with `clearTokenConfig`.

Client's commentary

> This is completely true in the case when a token is added to a platform, deleted and then added the second time. We will not add many tokens to a platform and we are not going to delete them from the Vault contract, instead we wil simply change their weight to 1 (as GMX did in the past).

H-2 High Incorrect usage of the Chainlink feed Acknowledged

Description

The `VaultPriceFeed.sol` contract receives the information about the price using the Chainlink feed.

From the Chainlink documentation the `latestAnsfer()` (https://docs.chain.link/data-feeds/api-reference#latestanswer) and `latestRound()` (https://docs.chain.link/data-feeds/api-reference#latestround) functions are deprecated.

However, these functions are used in the `VaultPriceFeed` contract:

VaultPriceFeed.sol#L281

VaultPriceFeed.sol#L302

VaultPriceFeed.sol#L309

Recommendation

We recommend using the Chainlink `latestRoundData()` function with safety checks and adding checks for the return data with proper revert messages if the price is stale or the round is incomplete.

Client's commentary

> This is completely true, Chainlink marked those functions as deprecated. The VaultPriceFeed contract has checks implemented which guarantee that execution continues only if latestRound() or latestAnswer() is not zero and the VaultPriceFeed contract can be changed.

H-3 High The protocol disadvantage in case of stable token depeg Acknowledged

Description

In the Vault, the `getPrice` function can return two price bounds – `upper` and `lower`. For stable tokens functions `getPriceV1` and `getPriceV2`, the return price is bound between the actual price and 1 USD (VaultPriceFeed.sol#L164).

```
function getPriceV1(address _token, bool _maximise, bool _includeAmmPrice) {
        uint256 price = getPrimaryPrice(_token, _maximise);

        ...
        if (strictStableTokens[_token]) {
            uint256 delta = price > ONE_USD ? price.sub(ONE_USD) : ONE_USD.
            if (delta <= maxStrictPriceDeviation) {
                return ONE_USD;
            }

            // if _maximise and price is e.g. 1.02, return 1.02
            if (_maximise && price > ONE_USD) {
                return price;
            }

            // if !_maximise and price is e.g. 0.98, return 0.98
            if (!_maximise && price < ONE_USD) {
                return price;
            }

            return ONE_USD;
        }
        ...
    }
```

If a stable token in the service balance is depegged downwards by 20%, then the `lower` price will be 0.8 USD and the `upper` price will be 1 USD for this token.

Then, in AUM calculation, if AUM is maximized, 1 USD will be used as the stable token price (NlpManager.sol#L136):

```
function getAum(bool maximise) public view returns (uint256) {
        uint256 length = vault.allWhitelistedTokensLength();
        uint256 aum = aumAddition;
        uint256 shortProfits = 0;
        IVault _vault = vault;

        for (uint256 i = 0; i < length; i++) {
            address token = vault.allWhitelistedTokens(i);
            ...

            uint256 price = maximise ? _vault.getMaxPrice(token) : _vault.g
            ...

            if (_vault.stableTokens(token)) {
                aum = aum.add(poolAmount.mul(price).div(10 ** decimals));
            } else {
                ...
            }
        }

        aum = shortProfits > aum ? 0 : aum.sub(shortProfits);
        return aumDeduction > aum ? 0 : aum.sub(aumDeduction);
    }
```

An unreasonably large AUM affects the NLP price and makes NLP tokens much more expensive to buy than they should be (NlpManager.sol#L209-L230):

```
function _addLiquidity(address _fundingAccount, address _account, address _
    ...

    // calculate aum before buyUSDG
    uint256 aumInUsdg = getAumInUsdg(true);
    ...

    // @audit AUM is much greater here then it should be, and mintAmount is
    uint256 mintAmount = aumInUsdg == 0 ? usdgAmount : usdgAmount.mul(glpSu
    ...

}
```

Then holders can sell it at a much lower price than it was bought (NlpManager.sol#L232-L260):

```
function _removeLiquidity(address _account, address _tokenOut, uint256 _glp
        ...

        // calculate aum before sellUSDG
        uint256 aumInUsdg = getAumInUsdg(false);
        uint256 glpSupply = IERC20(glp).totalSupply();

        // @audit low AUM is used here and sell price is low
        uint256 usdgAmount = _glpAmount.mul(aumInUsdg).div(glpSupply);
        uint256 usdgBalance = IERC20(usdg).balanceOf(address(this));
        if (usdgAmount > usdgBalance) {
            IUSDG(usdg).mint(address(this), usdgAmount.sub(usdgBalance));
        }

        IMintable(glp).burn(_account, _glpAmount);

        IERC20(usdg).transfer(address(vault), usdgAmount);
        uint256 amountOut = vault.sellUSDG(_tokenOut, _receiver);
        require(amountOut >= _minOut, "GlpManager: insufficient output");

        emit RemoveLiquidity(_account, _tokenOut, _glpAmount, aumInUsdg, gl

        return amountOut;
    }
```

This situation incentivizes users not to act and does not provide liquidity because any purchase transaction leads to money loss for the user, and the protocol is stopped.

Recommendation

Our recommendation is not to force the price of stablecoins to be 1 USD, but to take advantage of the market situation.

Client's commentary

> This is true, in cases where the price is minimized and the actual price is lower than 1 USD, the actual price will be returned. In cases where the price is maximized and the actual price is higher than 1 USD, the actual price will be returned. As noted, this is true if `delta > maxStrictPriceDeviation`. Currently, `maxStrictPriceDeviation` is set to `5000000000000000000000000000` (0.005 USD). If this behavior is needed to be changed, then editing the maxStrictPriceDeviation variable is sufficient.

Medium

M-1	Medium	A possibility of DoS (`allWhitelistedTokens`)	Acknowledged

Description

Limiting the maximum gas limit per block creates the possibility of a DoS that may occur because there is no limit on the `allWhitelistedTokens` from the `Vault` contract. The loop from the `NlpManager` iterates over the `allWhitelistedTokens`. If the count of the token is too big, the `getAum()` function from the `NlpManager` won't work.

Recommendation

We recommend adding a value for the maximum number of tokens which the government can add to the whitelist.

Client's commentary

> We are not going to add many tokens to a platform because the `allWhitelistedTokens` array can become unreasonably long, which will stop `NlpManager` from issuing nlp tokens and as a result it will stop everything on the platform, however, this exact issue can only occur with a high number of tokens added.

M-2 Medium Tokens cannot be removed from the whitelist Acknowledged

Description

There is an array of whitelisted tokens in the `Vault` contract. The governance can add tokens with the `setTokeConfig` function but can't remove them with the `clearTokenConfig` function because the governance contract `Timelock` doesn't have this function.

The functions listed below cannot be called from the `Timelock` contract:

- Vault.sol#L245
- Vault.sol#L259
- Vault.sol#L393
- NlpManager.sol#L93

Recommendation

We recommend adding the `vaultClearTokenConfig` function to the `Timelock` contract.

Client's commentary

> This is true, the Timelock contract does not have all the functionality. This can be easily bypassed by transferring gov of a needed contract to EOA.

M-3 Medium Users may not pay a deposit fee Acknowledged

Description

The `executeIncreasePosition` function in the `PositionRouter` contract uses the `_collectFees` function to calculate the fee amount. Since the first address is `msg.sender`, in case the Executor is calling the function, the fees will be equal to zero because the Executor won't have any position.
(PositionRouter.sol#L432)

Recommendation

We recommend changing the `msg.sender` to the `request.account`.

Client's commentary

> While true, the same error is present on GMX. This "deposit fee" that is taken during executeIncreasePosition is set to 0.3% (on GMX and on Nether.fi). Though it does not work properly, the execution fee that is taken in ETH works properly.

M-4 Medium **A possibility to remove liquidity without the cooldown duration** Acknowledged

Description

The `NlpManager` contract allows the minting and burning of NLP tokens and this contract has the cooldown duration after minting NLP tokens. The cooldown duration represents the time that needs to pass for the user before they can remove liquidity.

When the user withdraws liquidity, this contract validates that the cooldown duration has passed. However, the user can transfer the NLP tokens to another account and remove liquidity.

(NlpManager.sol#L234)

Recommendation

We recommend fixing it.

Client's commentary

> While true, it is unclear how this can be fixed, as the ERC20 tokens are "fungible". Furthermore, cooldownDuration is set to zero on GMX and on Nether.Fi.

M-5 Medium There are no limits for variables Acknowledged

Description

There are no limits to the following variables:

- `spreadThresholdBasisPoints` (VaultPriceFeed.sol#L120)
- `depositFee` (BasePositionManager.sol#L110)
- `aumAddition` and `aumDeduction` (NlpManager.sol#L93)
- `increasePositionBufferBps` (BasePositionManager.sol#L115)

Recommendation

We recommend adding variable ranges.

M-6 Medium Positions and token balances are lost in the `Vault` upgrade — Acknowledged

Description

The `Vault` upgrade mechanism allows one to migrate to another `Vault` by sending tokens.
(Vault.sol#L437-L440)

However, the `Vault` contract stores open positions that the governance will not transfer to a new `Vault`.
(Vault.sol#L128-L129)

Old users will not be able to close their positions opened in the old vault after migration.

Also, after calling the `upgradeVault` function in the `Vault` contract, the balance of the new `Vault` is the value transferred with the `upgrade` function, but the `tokenBalances` mapping in the new `Vault` contract is empty. It can't be set manually by governance.
(Vault.sol#L94)

Recommendation

We recommend implementing a complex way to migrate the `Vault` that doesn't drop positions.

Client's commentary

> True, right now it is unclear how to efficiently migrate `Vault` and it will require a lot of effort (and gas) to do this in the most efficient way.

M-7 Medium A fixed fee reward Acknowledged

Description

The `liquidatePosition` function in the `Vault` contract sends the fixed `liquidationFeeUsd` amount to the liquidator as the reward.
The liquidator can create a lot of small positions with amounts less than the reward and then liquidate them, receiving the fixed amount for each liquidation.

Vault.sol#L751

Recommendation:

We recommend adding a fee percentage for small positions.

Client's commentary

> True, however, `Vault` is now in a private liquidation mode and only liquidator is a position manager. There is a liquidator role on position manager, so presumably only a backend address is able to liquidate positions, users should be aware of this and not create many small positions.

M-8 Medium A possibility to lose access for governance Acknowledged

Description

The governance can set the `gov` address to a non-active address. If this happens, the governance loses access to the protocol.

In `Router.sol`:

Router.sol#L48

In `Vault.sol`:

Vault.sol#L294

In `VaultPriceFeed.sol`:

VaultPriceFeed.sol#L68

In `BaseToken.sol`:

BaseToken.sol#L52

In `YiledToken.sol`:

YieldToken.sol#L53

In `Governable.sol`:

Governable.sol#L17

In `OrderBook.sol`:

OrderBook.sol#L277

Recommendation:

We recommend adding a two-step procedure for `setGov` functions.

Client's commentary

> True, it can be done due to a human error, `Timelock` and/or other contracts can be changed.

M-9 Medium Access to user token transfers Acknowledged

Description

In the `YieldToken` contract, the governance can block token transfers:

```
function _transfer(address _sender, address _recipient, uint256 _amount) pr
    ...
    // @audit governance can disable token transferring
    if (inWhitelistMode) {
        require(whitelistedHandlers[msg.sender], "YieldToken: msg.sender no
    }
```

(YieldToken.sol#L185)

In the `BaseToken` contract, the governance can withdraw any user tokens:

```
function transferFrom(address _sender, address _recipient, uint256 _amount)
    // @audit Handler set by governance can withdraw any user tokens
    if (isHandler[msg.sender]) {
        _transfer(_sender, _recipient, _amount);
        return true;
    }
```

(BaseToken.sol#L144)

While in the `BaseToken` contract, the governance can block token transfers:

```
function _transfer(address _sender, address _recipient, uint256 _amount) pr
    ...

    // @audit Governance can block token transferring
    if (inPrivateTransferMode) {
        require(isHandler[msg.sender], "BaseToken: msg.sender not whitelist
    }
```

(BaseToken.sol#L188)

Recommendation

We recommend finding another solution for token transfers.

Client's commentary

> True. This is done to achieve better user experience with transferring many tokens on a platform.

M-10 Medium A problem with the access control Acknowledged

Description

After setting the new `gov` address, the old `gov` address has access to several functions as `admin`.

YieldToken.sol#L49

YieldToken.sol#L54

YieldToken.sol#L87

YieldToken.sol#L94

YieldToken.sol#L101

Recommendation

We recommended removing admin's rights from the old `gov` address.

Client's commentary

> True, however, admin was already removed from `USDG` (only the affected contract) from the deployer address. The deployer is not a gov nor admin on `USDG`. The gov address is only gov, not admin.

M-11 Medium **Incorrect liquidity adding when transferring a few tokens** Acknowledged

Description

The `addLiquidity` function in the `NlpManager` contract is incorrect when a few tokens are transferred.

A possible flow:

1. Add liquidity (e.g. 1000).
2. Remove part of the liquidity (999, but save 1). The total supply of NLP has to be 1.
3. Add liquidity again (e.g. 100). The tokens for deposit are transferred. NLP Tokens are not received.

Recommendation

We recommend adding the `_minGlp > 0` check or checking that the `mintAmount` is not zero.

Client's commentary

> True, however, the `_minUsdg` and `_minGlp` values are regulated on the frontend, so it is unlikely for an average user to encounter this problem.

M-12 | Medium | USDG tokens can be bought and sold in one transaction | Acknowledged

Description

The `buyUSDG` and `sellUSDG` functions in the `Vault` contract can be called within one transaction that allows price manipulation issues to be exploited using flash loans.

The `NLP` token that wraps `USDG` has protection from calling the `removeLiquidity` function in one transaction with the `addLiquidity` function.
(NlpManager.sol#L232-L235)

```
function _removeLiquidity(address _account, address _tokenOut, uint256 _glp
    // @audit in NLP token, it's prohibited to remove liquidity in the same
    require(lastAddedAt[_account].add(cooldownDuration) <= block.timestamp,
```

However, in the `sellUSDG` function in the `Vault` contract, there are no such checks.

Recommendation

We recommend adding checks for this case.

Client's commentary

> True, but the only address with access to those functions is NlpManager. Furthermore, cooldownDuration is set to zero on GMX and on Nether.Fi.

Low

L-1	Low	Naming duplication	Acknowledged

Description

There are many typos `glp` or `nlp` in the `NlpManager` contract.

Recommendation

It's recommended to fix all typos.

Client's commentary

> True, during renaming "glp" was changed to "nlp" where possible (where it would not affect frontend)

L-2 Low Old Solidity version and floating pragma Acknowledged

Description

Many contracts use the 0.6.12 version of the Solidity compiler. By mitigating the new version of the compiler, you lose new compiler optimizations and bug fixes, which can save gas and decrease the probability of attacks specific to old solidity versions.

Locking the pragma helps ensure that contracts do not accidentally get deployed using a different compiler version.
BasePositionManager.sol#L3
PositionManager.sol#L3
PositionRouter.sol#L3

Recommendation

We recommend updating contracts to the latest Solidity version and locking pragma to a specific Solidity version.

Client's commentary

> True, however, we did not change the solidity version to ensure that our contracts will follo the GMX contracts as closely as possible and to avoid potential errors. All currently deployed contracts were compiled using solidity version 0.6.12 with 1 optimizer runs.

L-3 Low Events for the setter functions Acknowledged

Description

Many functions have no events.

- Vault.sol#L279
- BaseToken.sol#L69
- YieldToken.sol#L62
- NlpManager.sol#L75
- VaultPriceFeed.sol#L72
- BasePositionManager.sol#L55
- PositionManager.sol#L27
- and other setters

Recommendation

We recommend adding events for the setters function for old and new values.

Client's commentary

> True, but it can still be tracked via onchain transactions.

L-4 Low Stucked ETH transfer out Acknowledged

Description

When the transaction is successful with increasing/decreasing a trader's position but the ETH transfer is unsuccessful and the ETH recipient will not receive his profit or execution fee: BasePositionManager.sol#L285

Stucked ETH can be withdrawn by admin or governance: BasePositionManager.sol#L153

Recommendation

We recommend checking the return value, and if the ETH transfer failed, then re-wrap it back to WETH and transfer it to the receiver.

Client's commentary

> True, however, it's only likely to occur if the receiver is a contract which does some action (potentially malicious) upon receiving ETH. It's done this way to ensure some level of safety.

L-5 Low Unsafe default price feed Acknowledged

Description

By default, the `Vault` contract and the `VaultPriceFeed` contract use params:

Vault.sol#L73

VaultPriceFeed.sol#L29-L31

```
contract Vault is ReentrancyGuard, IVault {
    ...
    bool public includeAmmPrice = true;
    ...
}

contract VaultPriceFeed is IVaultPriceFeed {
    ...

    bool public isAmmEnabled = true;
    bool public isSecondaryPriceEnabled = true;
    bool public useV2Pricing = false;
```

These flags enable unsafe V1 pricing that is vulnerable to oracle manipulation attack:

VaultPriceFeed.sol#L371

```
// if divByReserve0: calculate price as reserve1 / reserve0
// if !divByReserve1: calculate price as reserve0 / reserve1
function getPairPrice(address _pair, bool _divByReserve0) public view returns
    (uint256 reserve0, uint256 reserve1, ) = IPancakePair(_pair).getReserves
    if (_divByReserve0) {
        if (reserve0 == 0) { return 0; }
        return reserve1.mul(PRICE_PRECISION).div(reserve0);
    }
    if (reserve1 == 0) { return 0; }
    return reserve0.mul(PRICE_PRECISION).div(reserve1);
}
```

In this case, AMM is used, which is vulnerable to price manipulation (for example, changing reserves using a flash loan). This vulnerability can be used for sandwich attacks.

Recommendation

We recommend changing default values for the `isAmmEnabled`, `useV2Pricing` variables to `false`, in order to prohibit their changing to unsafe values or removing this code.

Client's commentary

> True, however, isAmmEnabled and useV2Pricing are already set to false. Furthermore, it is better to never use amm pricing.

L-6 Low Input value checks Acknowledged

Description

Some input params have no checks.

- `amount`:
  MintableBaseToken.sol#L24
  YieldToken.sol#L130
  NlpManager.sol#L216
- `liquidationFeeUsd`, `fundingRateFactor` and `stableFundingRateFactor`:
  Vault.sol#L235
- `_amountIn` and `_sizeDelta`:
  PositionManager.sol#L80
  Router.sol#L109

Recommendation

We recommend adding input values checks.

L-7 Low Chainlink flags Acknowledged

Description

In the `VaultPriceFeed` contract, the `getPrimaryPrice` function uses the Arbitrum Flag from Chainlink.

VaultPriceFeed.sol#L24

VaultPriceFeed.sol#L292

Recommendation

We recommend updated functions for the Base Network.

Client's commentary

> True, however, this is currently unused functionality and the chainlinkFlags contract can be set to a custom developed one.

L-8 Low The constructor creates two roles for one account Acknowledged

Description

The constructor creates two roles for one account:

YieldToken.sol#L4.

Recommendation

We recommend adding a variable for the admin role to input params.

L-9 Low No NatSpec Acknowledged

Description

The NatSpec is missing for all functions.

Recommendation

We recommend adding NatSpec.

L-10 Low Out of gas in possible infinite loops Acknowledged

Description

Some loops can make many iterations, leading to the out-of-gas situation.

BaseToken.sol#L101

BaseToken.sol#L108

BaseToken.sol#L218

YieldToken.sol#L102

YieldToken.sol#L109

YieldToken.sol#L215

NlpManager.sol#L142

Recommendation

We recommend setting the upper limit for `yieldTrackers` and validating it in the `setYieldTrackers` function in the `BaseToken` and `YieldToken` contracts.

Also, we recommend limiting the maximum number of tokens in the whitelist in the `setTokenConfig` function in the `Vault` contract.

Client's commentary

> True, however, every instance listed except for NlpManager is unused functionality.

L-11 Low No array length check Acknowledged

Description

A check that array lengths are the same is missing:

BasePositionManager.sol#L132

Recommendation

We recommend adding this check.

L-12 Low The allowance double-spend exploit Acknowledged

Description

Contracts are vulnerable to the allowance double-spend exploit.

- YieldToken.sol#L139
- BaseToken.sol#L138

Recommendation

We recommended implementing and using `safeIncreaseAllowance` and `safeDecreaseAllowance`.

L-13 Low The name and symbol of the token can be changed Acknowledged

Description

The `setInfo` function allows changing the token's `name` and `symbol`:

YieldToken.sol#L57

BaseToken.sol#L56

Recommendation

We recommended removing this function.

Client's commentary

> We don't plan to use this feature.

L-14 Low Possible overflow Acknowledged

Description

SafeMath is used but not everywhere.

- Vault.sol#L1012
- Vault.sol#L1032
- BasePositionManager.sol#L344
- PositionManager.sol#L310

Recommendation

We recommend using `SafeMath`.

L-15 Low Division by zero Acknowledged

Description

The `supply` variable may be zero.

NlpManager.sol#L121

Recommendation

We recommend adding zero checks for the `supply` value.

Client's commentary

L-16 Low A Base network compatibility issue Acknowledged

Description

The protocol has native support for BNB and wBTC tokens:
(VaultPriceFeed.sol#L103-L107)

```
function setTokens(address _btc, address _eth, address _bnb) external onlyG
    btc = _btc;
    eth = _eth;
    bnb = _bnb;
}

function setPairs(address _bnbBusd, address _ethBnb, address _btcBnb) exter
    bnbBusd = _bnbBusd;
    ethBnb = _ethBnb;
    btcBnb = _btcBnb;
}
```

However, the `Base Network` has no support for these tokens.

Recommendation

We recommend removing unsupported tokens or reorganizing logic based on assets supported by the network.

Conclusion

During the audit process 3 HIGH, 12 MEDIUM, and 16 LOW severity findings have been spotted.

Disclaimer

The Stronghold audit makes no statements or warranties about the utility of the code, the safety of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug-free status. The audit documentation is for discussion purposes only.