Security Audit Report

YIN Protocol

SECBIT

November 22, 2021

1. Introduction

YIN is a decentralized finance protocol that provides a multi-strategy liquidity optimizer. SECBIT Labs conducted an audit from June 25th to November 22th, 2021, including an analysis of the contract in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the YIN Protocol contract 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
Design & Implementation	4.3.1 Possible flash loan attack vector exists in `unsubscribe()` function.	Medium	Fixed
Design & Implementation	4.3.2 Users may lose unused funds when subscribing.	Medium	Fixed
Design & Implementation	4.3.3 Transferring a Yang NFT token to a held address will result in a `_userMap` mapping error.	Low	Fixed
Gas Optimization	4.3.4 Redundant data structures can increase gas consumption.	Info	Fixed
Design & Implementation	4.3.5 A design flaw in the time lock resulted in users being able to bypass restrictions.	Medium	Fixed
Design & Implementation	4.3.6 Separate time locks should be designed for different `CHIVault` to avoid interactions.	Info	Fixed

Design & Implementation	4.3.7 Modify `div` to `mul` to fix logic error.	Low	Fixed
Design & Implementation	4.3.8 In the `CHIDepositCallback()` function, it may failed to transfer token by `transferFrom()` function.	Medium	Fixed
Gas Optimization	4.3.9 Redundant code can increase gas consumption at deployment.	Info	Fixed
Design & Implementation	4.3.10 Add a judgment on the amount of liquidity funds to be retrieved to prevent confusion in managing funds.	Info	Fixed
Design & Implementation	4.3.11 In the `YangNFTVault` contract, the wrong logic in the `subscribe()` function to handle the transfer of funds from the user can raise a risk to the security of the funds.	High	Fixed
Design & Implementation	4.3.12 In the `YangNFTVault` contract, there is a logical error in the `YANGDepositCallback()` function that causes confusion in the transfer of funds.	Medium	Fixed
Design & Implementation	4.3.13 In the `YangNFTVault` contract, the parameter `tokenOut` is incorrectly set to a value that prevents the user from withdrawing the destination asset.	Medium	Fixed
Design & Implementation	4.3.14 In the `YangNFTVault` contract, it is not necessary to	Info	Fixed

	grant access to user funds to the `CHIManager` contract by `safeApprove()` function.		
Design & Implementation	4.3.15 In the `CHIVault` contract, change the name of the `_collet()` function to `_collect()` to better match the content of the function.	Info	Fixed
Design & Implementation	4.3.16 In the `CHIManager` contract, some system parameters such as `manager` cannot be changed after they have been set.	Info	Fixed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about the YIN Protocol contract is shown below:

- Project website
  - https://yin.finance/
- Smart contract code
  - https://github.com/YinFinance/yin-core
  - initial review commit *d611388*
  - final review commit *7f50eff*

2.2 Contract List

The following content shows the contracts included in the YIN Protocol project, which the SECBIT team audits:

Name	Lines	Description
CHIManager.sol	551	A contract is used to manage CHI vaults.
CHIVault.sol	870	A single strategy contract and users can subscribe CHI to transform their assets into liquidity.
CHIVaultDeployer.sol	28	An auxiliary contract is used to create new `CHIVault` contract.

ICHIDepositCallBack.sol	11	An interface contract that provides `CHIDepositCallback` function.
ICHIManager.sol	219	Provides the interface for the `CHIManager` contract functions.
ICHIVault.sol	101	Provides the interface for the `CHIVault` contract function.
ICHIVaultDeployer.sol	17	Provide the interface for the `CHIVaultDeployer` contract functions.
IChainLinkFeedsRegistry.sol	20	Provides interface to obtain the current asset price.
IOracleProvider.sol	31	Provides the interface for the `OracleProvider` contract function.
IYANGDepositCallBack.sol	11	An interface contract that provides `YANGDepositCallback` function.
IYangNFTVault.sol	79	Provides the interface for the `YangNFTVault` contract function.
LiquidityHelper.sol	31	An auxiliary contract used to manage `CHIVault` liquidity.
PriceHelper.sol	29	An auxiliary contract that calculates the current value of `CHIVault` assets.
YANGPosition.sol	13	An auxiliary contract that provides position calculation.
YINToken.sol	24	Standard ERC20(YIN) token contract.

OracleProvider.sol	94	A oracle contract that provides price data.
Timelock.sol	173	Add time-locked contract to transactions.
ChainLinkFeedsRegistry.sol	83	Provide functions to obtain the current asset price.
LockLiquidity.sol	51	A liquidity locking auxiliary contract.
YangNFTVault.sol	271	All subsequent asset management actions and processes will be performed from this contract.

3. Contract Analysis

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

3.1 Role Classification

The protocol has three key roles: Governance Account, Strategy Provider Account, and Common Account.

- Governance Account
  - Description

    Contract administrator
  - Authority
    - Update basic parameters
    - Transfer ownership
  - Method of Authorization

The contract administrator is the creator of the contract or authorized by the transferring of governance account.

- Strategy Provider Account
  - Description

    The creator of the CHI NFT token
  - Authority
    - Mint CHI NFT token
    - Manager liquidity
  - Method of Authorization

    The strategy provider account is granted by governance.
- Common Account
  - Description

    Users participating in YIN Protocol
  - Authority
    - Mint Yang NFT token
    - choose to subscribe or unsubscribe from various CHI vault
  - Method of Authorization

    No authorization required

3.2 Functional Analysis

The YIN Protocol provides users with automatic reinvestment and active liquidity management strategies. 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 several parts.

CHIManager

The `CHIManager` contract serves as a bridge between the `YangNFTVault` contract and the `CHIVault` contract. In addition, it takes on the task of creating new vaults.

The main functions in `CHIManager` are as below:

- `mint()`

  Strategy providers can design their own liquidity strategies and mint CHI NFT token.

- `subscribe()`

  This function transfers the funds subscribed by the user under the `YangNFTVault` contract to the corresponding `CHIVault` contract.

- `unsubscribe()`

  In contrast to the `subscribe()` function, this function is used to withdraw the user's subscription funds and earnings.

- `addTickPercents()`

  This function is used to modify the percentage of market-making funds that are allocated to each tick in the specified `CHIVault`.

- `addRange()`

  This function is used to add a new tick interval.

- `removeRange()`

  This function is used to remove a tick interval.

- `addAllLiquidityToPosition()`

  This function transfers funds from the `CHIVault` contract to Uniswap v3 for market making.

- `removeAllLiquidityFromPosition()`

  This function removes all market-making funds and earnings for the specified tick.

- `addLiquidityToPosition()`

  This function adds liquidity to the specified tick.

CHIVault

The strategy provider would create each liquidity smart contract as a CHI, an NFT token following the ERC-721 Standard. Managing funds in Yang smart vault through the use of CHI. The functions in this contract are specific implementations of the `CHIManager` contract functions.

The main functions in `CHIVault` are as below:

- `getTotalLiquidityAmounts()`

  This function returns all market-making funds and earnings in the current `CHIVault` contract.

- `getTotalAmounts()`

  This function returns the principal and earnings belonging to the user.

- `deposit()`

  This function transfers the funds from the `YangNFTVault` contract to the `CHIVault` contract and updates the shares.

- `withdraw()`

  This function calculates the corresponding withdrawal funds based on shares and sends the funds to the `yangNFTvault` contract address.

- `swapPercentage()`

  Exchange the token under this contract for another token on a proportional basis.

YangNFTVault

Users can use this contract to create their own Yang NFT token, which will be used to manage their funds.

The main functions in `YangNFTVault` are as below:

- `mint()`

  Any user can use this function to mint a Yang NFT token, but only one address can hold.

- `subscribe()`

  User deposits funds and subscribes to shares according to this function.

- `unsubscribe()`

  Users can withdraw the specified share of funds according to this function. The retrieved funds include not only the principal but also the earnings.

4. Audit Detail

This part describes the process, and the detailed results of the audit 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 bug, 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 two 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 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 Possible flash loan attack vector exists in `unsubscribe()` function.

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

Description

Malicious users can receive the rewards of market-making without actually participating in liquidity market-making (and without actually contributing to the protocol), which affects the earnings of normal contract participants and breaks the normal use of the contract.

```
// @audit located on YangNFTVault.sol
function unsubscribe(IYangNFTVault.UnSubscribeParam memory
params)
        external
        override
        nonReentrant
        isAuthorizedForToken(params.yangId)
    {
        ......

        (uint256 amount0, uint256 amount1) =
ICHIManager(chiManager).unsubscribe(
            params.yangId,
            params.chiId,
            params.shares,
            params.amount0Min,
            params.amount1Min
```

```
        );

        ......
    }

// @audit located on CHIManager.sol
function unsubscribe(
        uint256 yangId,
        uint256 tokenId,
        uint256 shares,
        uint256 amount0Min,
        uint256 amount1Min
    )
        external
        override
        onlyYANG
        nonReentrant
        returns (uint256 amount0, uint256 amount1)
    {
        ......

        (amount0, amount1) = ICHIVault(_chi_.vault).withdraw(
            yangId,
            shares,
            amount0Min,
            amount1Min,
            yangNFT
        );

        _position.shares =
positions[positionKey].shares.sub(shares);
    }
```

Status

The development team added a time lock and fixed this issue in commit 9d74258.

4.3.2 Users may lose unused funds when subscribing.

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

Description

The `subscribe()` function is called when a user subscribes to shares, which calls the internal function `deposit()`. At this point, the user transfers the desired principal `amount0Desired` and `amount1Desired` to the `YangNFTVault` contract. However, depending on the actual situation, some of the funds may not be available for subscription. Then the excess funds will be left in the `YangNFTVault` contract, and the user will lose these funds.

```
function deposit(
        address token0,
        uint256 amount0,
        address token1,
        uint256 amount1
    ) internal {
        require(amount0 > 0, 'NZ');
        require(amount1 > 0, 'NZ');
        IERC20(token0).safeTransferFrom(msg.sender,
address(this), amount0);
        IERC20(token1).safeTransferFrom(msg.sender,
address(this), amount1);
    }

......

function _subscribe(
        address token0,
        address token1,
        IYangNFTVault.SubscribeParam memory params
    )
        internal
```

```
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 shares
        )
    {
        IERC20(token0).safeApprove(chiManager,
params.amount0Desired);
        IERC20(token1).safeApprove(chiManager,
params.amount1Desired);

        (shares, amount0, amount1) =
ICHIManager(chiManager).subscribe(
            params.yangId,
            params.chiId,
            params.amount0Desired,
            params.amount1Desired,
            params.amount0Min,
            params.amount1Min
        );

        if (params.amount0Desired - amount0 > 0) {
            bytes32 key0 =
keccak256(abi.encodePacked(params.yangId, token0));
            _vaults[key0] = _vaults[key0].add(amount0);
        }

        if (params.amount1Desired - amount1 > 0) {
            bytes32 key1 =
keccak256(abi.encodePacked(params.yangId, token1));
            _vaults[key1] = _vaults[key1].add(amount1);
        }

        IERC20(token0).safeApprove(chiManager, 0);
        IERC20(token1).safeApprove(chiManager, 0);
    }


function subscribe(IYangNFTVault.SubscribeParam memory params)
```

```
        external
        override
        isAuthorizedForToken(params.yangId)
        nonReentrant
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 shares
        )
    {
        ......

        // deposit valut to yangNFT and then to chi
        deposit(token0, params.amount0Desired, token1,
params.amount1Desired);

        (amount0, amount1, shares) = _subscribe(token0,
token1, params);

        YangPosition.Info storage position =
_positions.get(params.yangId, params.chiId);
        position.amount0 = position.amount0.add(amount0);
        position.amount1 = position.amount1.add(amount1);
        position.shares = position.shares.add(shares);

        ......
    }
```

Status

The development team updated the code design structure and fixed this issue in commit 9d74258.

4.3.3 Transferring a Yang NFT token to a held address will result in a `_userMap` mapping error.

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

Description

Any user can call the `mint()` function to obtain a Yang NFT Token. The corresponding information will be recorded in `_usersMap`. Considering that one can transfer the Yang NFT token between different addresses, the `_usersMap` does not modify the recorded information, which will cause confusion in the recorded information and affect the normal operation of the contract.

```
// @audit located on YangNFTVault
mapping(address => uint256) private _usersMap;

function mint(address recipient) external override returns
(uint256 tokenId) {
        require(_usersMap[recipient] == 0, 'only mint once');
        // _mint function check tokenId existence
        _mint(recipient, (tokenId = _nextId++));
        _usersMap[recipient] = tokenId;

        emit MintYangNFT(recipient, tokenId);
 }

function getTokenId(address recipient) public view override
returns (uint256) {
        return _usersMap[recipient];
}
```

Suggestion

Add _beforeTokenTransfer() function to modify `_usersMap` mapping when Yang NFT token address is changed.

Status

The development team adopts our advice and fixes this issue in commit f5f14d0.

4.3.4 Redundant data structures can increase gas consumption.

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

Description

The `_positions` mapping is used to record the shares subscribed by the specified user. When a user calls the `unsubscribe()` function to retrieve the specified shares of funds, the current user's shares are checked to ensure enough shares. The share check operation is repeated under the `YangNFTVault` contract and the `CHIManager` contract, increasing gas consumption.

```
//@audit located on YANGPosition
mapping(bytes32 => YANGPosition.Info) private _positions;

//@audit located on YangNFTVault
function unsubscribe(UnSubscribeParam memory params)
        external
        override
        nonReentrant
        isAuthorizedForToken(params.yangId)
        afterLockUnsubscribe(msg.sender)
    {
        require(chiManager != address(0), 'CHI');
```

```
        (, , address _pool, , , , , ) =
ICHIManager(chiManager).chi(params.chiId);

        bytes32 key =
keccak256(abi.encodePacked(params.yangId, params.chiId));
        // @audit redundant check
        require(_positions[key].shares >= params.shares,
'insufficient shares');

      ......
    }
```

```
//@audit located on CHIManager
function unsubscribe(
        uint256 yangId,
        uint256 tokenId,
        uint256 shares,
        uint256 amount0Min,
        uint256 amount1Min
    )
        external
        override
        onlyYANG
        nonReentrant
        returns (uint256 amount0, uint256 amount1)
    {
        CHIData storage _chi_ = _chi[tokenId];
        require(!_chi_.config.archived, "CHI Archived");

        bytes32 positionKey =
keccak256(abi.encodePacked(yangId, tokenId));
        YANGPosition.Info storage _position =
positions[positionKey];

        //@audit first check
        require(_position.shares >= shares, "s");

      ......
    }
```

Suggestion

Remove the redundant check from the `YangNFTVault` contract to save gas.

Status

The development team adopts our advice and fixes this issue in commit f5f14d0.

4.3.5 A design flaw in the time lock resulted in users being able to bypass restrictions.

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

Description

A time lock is used in the `subscribe()` function to prevent flash loan attacks. Time lock is designed to target user addresses. It does not allow frequent operations on the same user address. The problem with this design mechanism is that the user can bypass the check by transferring the Yang NFT token to a new address for an attack.

```
//@audit loacted on YangNFTVault
function subscribe(SubscribeParam memory params)
        external
        override
        isAuthorizedForToken(params.yangId)
        nonReentrant
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 shares
        )
    {
        ...
```

```
        ...
        //@audit add time lock
        _updateAccountLockDurations(msg.sender,
block.timestamp);

        ...
    }

function _updateAccountLockDurations(address account, uint256
currentTime) internal {
        if (_isLocked) {
            uint256 durationTime =
currentTime.add(_lockInSeconds);
            _locks[account] = durationTime > _locks[account] ?
durationTime : _locks[account];
            emit LockAccount(account, _locks[account]);
        }
    }
```

Suggestion

Adding a time lock on the Yang NFT token can effectively avoid the above problem.

Status

The development adopts our advice and fixes this issue in commit f5f14d0.

4.3.6 Separate time locks should be designed for different `CHIVault` to avoid interactions.

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

Description

Users may subscribe to different `CHIVault` using the same Yang NFT token, so the design should be made so that each `CHIVault` has a separate time lock.

```
function _updateAccountLockDurations(address account, uint256
currentTime) internal {
        if (_isLocked) {
            uint256 durationTime =
currentTime.add(_lockInSeconds);
            _locks[account] = durationTime > _locks[account] ?
durationTime : _locks[account];
            emit LockAccount(account, _locks[account]);
        }
    }
```

Status

The development team adopts our advice and fixes this issue in commit f5f14d0.

4.3.7 Modify `div` to `mul` to fix logic error.

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

Description

Mistakenly writing `mul` as `div` in the `collectProtocol()` function will cause a `providerFee` calculation error.

```
// @audit located on CHIManager
function collectProtocol(uint256 tokenId) external override {
        ......
        uint256 amount0 =
accruedProtocolFees0.mul(_providerFee).div(1e6);
        // @audit use mul instead of div
        uint256 amount1 =
accruedProtocolFees1.div(_providerFee).div(1e6);
}
```

Status

The development team fixes this issue in commit fa54399.

4.3.8 In the `CHIDepositCallback()` function, it may failed to transfer token by `transferFrom()` function.

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

Description

Due to the history of solidity upgrades, transferring token using the `IERC20(token).transferFrom()` method will fail for some specific ERC20 tokens.

```
    function CHIDepositCallback(
        IERC20 token0,
        uint256 amount0,
        IERC20 token1,
        uint256 amount1
    ) external override {
        _verifyCallback(msg.sender);

        if (amount0 > 0) token0.transferFrom(yangNFT,
msg.sender, amount0);
        if (amount1 > 0) token1.transferFrom(yangNFT,
msg.sender, amount1);
    }
```

Suggestion

The recommended modification is to use the `SafeREC20` library function instead of the `ERC20` library, see : SafeERC20.sol.

Status

The development team fixes this issue in commit fa54399.

4.3.9 Redundant code can increase gas consumption at deployment.

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

Description

The following three functions in the `YangNFTVault.sol` contract do not appear to be used in the current contract. Since these three functions are declared as `internal` or `private` types, they cannot be used by any interface either. Additional gas will be consumed when the contract is deployed, so we recommend removing them to save gas if these three functions are not an additional consideration.

```
// @aduit located on YangNFTVault
function _amountsForLiquidity(
    IUniswapV3Pool pool,
    int24 tickLower,
    int24 tickUpper,
    uint128 liquidity
) internal view returns (uint256 amount0, uint256 amount1) {
    ......
}

function _liquidityForAmounts(
    IUniswapV3Pool pool,
    int24 tickLower,
    int24 tickUpper,
    uint256 amount0,
    uint256 amount1
) internal view returns (uint128 liquidity) {
    ......
}

function toUint128(uint256 x) private pure returns (uint128 y)
{
    require((y = uint128(x)) == x);
    }
```

Status

The development team adopts our advice and fixes this issue in commit 159b117.

4.3.10 Add a judgment on the amount of liquidity funds to be retrieved to prevent confusion in managing funds.

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

Description

The `withdraw()` function in the `CHIVault` contract retrieves the principal and earnings deposited by the user for the specified shares. When the remaining funds under the `CHIVault` contract are not enough to cover the number of funds retrieved by the user, the contract will retrieve a portion of the liquidity funds to cover the shortfall.

To be on the safe side, when using the `_burnMultLiquidityScale()` function to remove the corresponding liquidity and retrieve funds, it is advisable to determine the number of funds to be retrieved to prevent the use of the agreement fee due to insufficient funds being retrieved, which may lead to confusion in funds management.

```
function _withdrawShare(
        uint256 shares,
        uint256 amount0Min,
        uint256 amount1Min
    ) internal returns (uint256 withdrawal0, uint256
withdrawal1) {
       ......
            // avoid round down
            shouldLiquidity =
shouldLiquidity.mul(10100).div(10000);
            if (shouldLiquidity > scaleRate) {
                shouldLiquidity = scaleRate;
            }

            _burnMultLiquidityScale(shouldLiquidity,
address(this));
        }
        // Burn shares
        _burn(shares);
```

```
    }
```

Suggestion

It is recommended to add the following judgment conditions. A refined method is as follows:

```
function _withdrawShare(
        uint256 shares,
        uint256 amount0Min,
        uint256 amount1Min
    ) internal returns (uint256 withdrawal0, uint256
withdrawal1) {
       ......
            // avoid round down
            shouldLiquidity =
shouldLiquidity.mul(10100).div(10000);
            if (shouldLiquidity > scaleRate) {
                shouldLiquidity = scaleRate;
            }

            // @audit add following codes
      (total0, total1) =
_burnMultLiquidityScale(shouldLiquidity, address(this));
      require(total0 >= shouldWithdrawal0 && total1 >=
 shouldWithdrawal1, "SW");
    }
        }
        // Burn shares
        _burn(shares);
    }
```

Status

The development team adopts our advice and fixes this issue in commit 159b117.

4.3.11 In the `YangNFTVault` contract, wrong logic in the `subscribe()` function to handle the transfer of funds from the user can raise a risk to the security of the funds.

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

Description

The user can subscribe to shares using the `subscribe()` function and must provide both `token0` and `token1` assets. The structure `SubscribeParam` records the user's subscription information. The parameters `amount0Desired` and `amount1Desired` represent the desired amount of `token0` and `token1` assets to be subscribed by the user. The internal function `_subscribe()` transfers assets from a user's external account to `CHIVault` contract. The specific operations implemented within the `_subscribe()` function can be split into three steps as follows:

(1). First, the `subscribe()` function is called under the `CHIManager` contract. It completes the task of linking the `YangNFTVault` contract to the `CHIVault` contract.

(2). Secondly, the `subscribe()` function under the `CHIManager` contract will call the `deposit()` function under the `CHIVault` contract, which will calculate the amount of `token0` and `token1` assets, and the corresponding `shares` that the user should deposit under the `CHIVault` contract.

(3). Finally, the `deposit()` function under the `CHIVault` contract will call the `CHIDepositCallback()` function, which will call `YANGDepositCallback()` function under the `YangNFTVault` contract. It will transfer the user's `token0` and `token1` assets to the `CHIVault` contract, and complete the entire subscription operation.

A detailed analysis of the code shows that the user does not transfer `amount0Desired` amount of `token0` assets and `amount1Desired` amount of `token1` assets to the `YangNFTVault` contract, but rather the actual number of shares subscribed. Therefore, the logic of the `subscribe()` function under the `YangNFTVault` contract to handle excess transfers is incorrect. It may cause a severe risk to the security of the funds.

```
struct SubscribeParam {
        uint256 yangId;
        uint256 chiId;
        uint256 amount0Desired;
        uint256 amount1Desired;
        uint256 amount0Min;
        uint256 amount1Min;
    }

//@audit located on YangNFTVault
function subscribe(SubscribeParam memory params)
        external
        override
        subscripting(msg.sender)
        isAuthorizedForToken(params.yangId)
        nonReentrant
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 shares
        )
    {
        ......

        (amount0, amount1, shares) = _subscribe(token0,
token1, params);
        ......

        // _withdraw rest to msg.sender
        uint256 remain0 = params.amount0Desired.sub(amount0);
        uint256 remain1 = params.amount1Desired.sub(amount1);
```

```
        _withdraw(token0, remain0, token1, remain1);

        ......
    }

//@audit located on YangNFTVault
function _subscribe(
        address token0,
        address token1,
        SubscribeParam memory params
    )
        internal
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 shares
        )
    {
        ......

        (shares, amount0, amount1) =
ICHIManager(chiManager).subscribe(
            params.yangId,
            params.chiId,
            params.amount0Desired,
            params.amount1Desired,
            params.amount0Min,
            params.amount1Min
        );

       ......
    }

//@audit located on CHIManager
function subscribe(
        uint256 yangId,
        uint256 tokenId,
        uint256 amount0Desired,
        uint256 amount1Desired,
```

```
        uint256 amount0Min,
        uint256 amount1Min
    )
        ......
    {
        (shares, amount0, amount1) =
ICHIVault(_tempVault).deposit(
            yangId,
            amount0Desired,
            amount1Desired,
            amount0Min,
            amount1Min
        );

        ......
        );
    }

//@audit located on CHIVault
function deposit(
        uint256 yangId,
        uint256 amount0Desired,
        uint256 amount1Desired,
        uint256 amount0Min,
        uint256 amount1Min
    )
        external
        override
        nonReentrant
        onlyManager
        returns (
            uint256 shares,
            uint256 amount0,
            uint256 amount1
        )
    {
        ......
        (shares, amount0, amount1) = _calcSharesAndAmounts(
            total0,
```

```
            total1,
            amount0Desired,
            amount1Desired
        );
        ......

        // Pull in tokens from sender
        ICHIDepositCallBack(msg.sender).CHIDepositCallback(
            token0,
            amount0,
            token1,
            amount1,
            address(this)
        );

        ......
    }

//@audit located on CHIManager
function CHIDepositCallback(
        IERC20 token0,
        uint256 amount0,
        IERC20 token1,
        uint256 amount1,
        address recipient
    ) external override {
        _verifyCallback(msg.sender);
        IYANGDepositCallBack(yangNFT).YANGDepositCallback(
            token0,
            amount0,
            token1,
            amount1,
            recipient
        );
    }

//@audit located on YangNFTVault
function YANGDepositCallback(
        IERC20 token0,
```

```
        uint256 amount0,
        IERC20 token1,
        uint256 amount1,
        address recipient
    ) external override
    {
        require(chiManager != address(0), "CHI");
        require(msg.sender == chiManager, "manager");
        if (amount0 > 0) token0.safeTransferFrom(_tempAccount,
recipient, amount0);
        if (amount1 > 0) token1.safeTransferFrom(_tempAccount,
recipient, amount0);
    }
```

Suggestion

Remove the funds processing code from the `subscribe()` function under the `YangNFTVault` contract. The recommended change is as follows:

```
function subscribe(SubscribeParam memory params)
        external
        override
        subscripting(msg.sender)
        isAuthorizedForToken(params.yangId)
        nonReentrant
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 shares
        )
    {
        require(chiManager != address(0), 'CHI');
        (, , address _pool, , , , , ) =
ICHIManager(chiManager).chi(params.chiId);
        IUniswapV3Pool pool = IUniswapV3Pool(_pool);
        address token0 = pool.token0();
        address token1 = pool.token1();
        require(!checkMaxUSDLimit(params.chiId), 'MUL');
```

```
        (amount0, amount1, shares) = _subscribe(token0,
token1, params);
        _updateAccountLockDurations(params.yangId,
params.chiId, block.timestamp);

        //@audit delete the following codes
        // _withdraw rest to msg.sender
        //uint256 remain0 =
params.amount0Desired.sub(amount0);
        //uint256 remain1 =
params.amount1Desired.sub(amount1);
        //_withdraw(token0, remain0, token1, remain1);

        emit Subscribe(params.yangId, params.chiId, shares);
    }
```

Status

The development team adopts our advice and fixes this issue in commit 159b117.

4.3.12 In the `YangNFTVault` contract, there is a logical error in the `YANGDepositCallback()` function that causes confusion in the transfer of funds.

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

Description

The `YANGDepositCallback()` function implements the fund transfer capability under the `YangNFTVault` contract. With the condition `amount1 > 0`, an amount of `amount1` should be transferred to the `recipient` address.

```
function YANGDepositCallback(
        IERC20 token0,
```

```
        uint256 amount0,
        IERC20 token1,
        uint256 amount1,
        address recipient
    ) external override
    {
        require(chiManager != address(0), "CHI");
        require(msg.sender == chiManager, "manager");
        if (amount0 > 0) token0.safeTransferFrom(_tempAccount,
recipient, amount0);

        // @audit the number of funds transferred should be
amount1
        if (amount1 > 0) token1.safeTransferFrom(_tempAccount,
recipient, amount0);
    }
```

Suggestion

The recommended modification is as follows:

```
function YANGDepositCallback(
        IERC20 token0,
        uint256 amount0,
        IERC20 token1,
        uint256 amount1,
        address recipient
    ) external override
    {
        ......

        // @audit the modification is as follows
        if (amount1 > 0) token1.safeTransferFrom(_tempAccount,
recipient, amount1);
    }
```

Status

The development team adopts our advice and fixes this issue in commit 159b117.

4.3.13 In the `YangNFTVault` contract, the parameter `tokenOut` is incorrectly set to a value that prevents the user from withdrawing the destination asset.

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

Description

The `unsubscribeSingle()` function is used to retrieve single-sided assets and proceeds. This function calculates the amount of `token0` and `token1` assets based on the user's share retrieval. It then uses the `swap()` function in Uniswap V3 to convert all `token0` (`token1`) to `token1` (`token0`) depending on the type of single-sided assets the user wants to retrieve. The `params.zeroForOne` parameter distinguishes the kind of assets the user is retrieving. If it is `true`, then the user only wants to retrieve `token1` assets, all `token0` assets in the `shares` need to be exchanged for `token1` assets via Uniswap v3. All `token1` assets will then be sent to the user.

In the current code, when `params.zeroForOne` is true, the user will get `token0` asset, which is not consistent with the design and does not achieve the function.

```
function unsubscribeSingle(UnSubscribeSingleParam memory
params)
        external
        override
        nonReentrant
        isAuthorizedForToken(params.yangId)
        afterLockUnsubscribe(params.yangId, params.chiId)
    {
        ......
        address tokenOut = params.zeroForOne ? pool.token0() :
pool.token1();
        _withdraw(tokenOut, amount, address(0), 0);

        emit UnSubscribe(params.yangId, params.chiId, amount,
0);
    }
```

Suggestion

Adjust the corresponding value of the `tokenOut` parameter. The recommended modification is as follows:

```
function unsubscribeSingle(UnSubscribeSingleParam memory
params)
        external
        override
        nonReentrant
        isAuthorizedForToken(params.yangId)
        afterLockUnsubscribe(params.yangId, params.chiId)
    {
        ......
        // @audit fixed
        address tokenOut = params.zeroForOne ? pool.token1() :
pool.token0();
        _withdraw(tokenOut, amount, address(0), 0);

        emit UnSubscribe(params.yangId, params.chiId, amount,
0);
    }
```

Status

The development team adopts our advice and fixes this issue in commit 159b117.

4.3.14 In the `YangNFTVault` contract, it is not necessary to grant access to user funds to the `CHIManager` contract by `safeApprove()` function.

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

Description

The `CHIManager` contract in the Yin finance project does not perform funds transfer functions, so there is no need to authorize the `CHIManager` contract with a credit line. It is recommended that this part of the operation be removed.

```
function _subscribe(
        address token0,
        address token1,
        SubscribeParam memory params
    )
        internal
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 shares
        )
    {
        IERC20(token0).safeApprove(chiManager,
params.amount0Desired);
        IERC20(token1).safeApprove(chiManager,
params.amount1Desired);
```

```
        ......

        IERC20(token0).safeApprove(chiManager, 0);
        IERC20(token1).safeApprove(chiManager, 0);
    }


function _subscribeSingle(address tokenIn,
SubscribeSingleParam memory params)
        internal
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 shares
        )
    {
        IERC20(tokenIn).safeApprove(chiManager,
params.maxTokenAmount);

        ......

        IERC20(tokenIn).safeApprove(chiManager, 0);
    }
```

Status

The development team adopts our advice and fixes this issue in commit 159b117.

4.3.15 In the `CHIVault` contract, change the name of the `_collet()` function to `_collect()` to better match the content of the function.

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

Description

The function `_collet()` is used to retrieve a proceeds, rename it to `_collect()` to better fit the function.

```
function _harvestFee() internal {
        ......
        for (uint256 i = 0; i < _rangeSet.length(); i++) {
            ......
            (uint256 _collect0, uint256 _collect1) = _collet(
                _tickLower,
                _tickUpper
            );
            ......
    }

function _collet(int24 tickLower, int24 tickUpper)
        internal
        returns (uint256 collect0, uint256 collect1)
    {
        ......
    }
```

Status

The development team adopts our advice and fixes this issue in commit 159b117.

4.3.16 In the `CHIManager` contract, some system parameters such as `manager` cannot be changed after they have been set.

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

Description

Some system parameters in the `CHIManager` contract cannot be changed after the `initialize()` function is set, which may affect the robustness of the code. If the relevant parameters are set incorrectly, upgrading the contract or replacing it is necessary, which is costly.

Therefore, it is recommended that add an interface for the modification of relevant system parameters.

```
address public manager;
address public executor;
address public deployer;
address public treasury;
address public governance;

function initialize(
        bytes32 _merkleRoot,
        address _v3Factory,
        address _yangNFT,
        address _deployer,
        address _executor,
        address _manager,
        address _governance,
        address _treasury
    ) public initializer {
        ......

        manager = _manager;
        treasury = _treasury;
        governance = _governance;
        deployer = _deployer;
        executor = _executor;

        ......
    }
```

Status

The development team has clarified the meaning of the system parameters and added the parameter modification function in commit 159b117.

5. Conclusion

After auditing and analyzing the YIN Protocol contract, SECBIT Labs found some issues to optimize and proposed corresponding suggestions, which have been shown above. SECBIT Labs holds the view that the YIN Protocol contract has high code quality, concise implementation, and detailed documentation. The issues raised in the report above have all been discussed or fixed.

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 ethers 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