verichains

SECURITY AUDIT OF

METAIN REIT SMART CONTRACTS

METAIN
CO-INVESTMENT PLATFORM

Public Report

Oct 17, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Oct 17, 2022. We would like to thank the METAIN for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the METAIN REIT Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code. METAIN fixed the code, according to Verichains's draft report

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About METAIN REIT Smart Contracts

METAIN is the blockchain-empowered co-investment platform focusing in real estate, whose vision is to provide high-yield, transparent and secured investment opportunities to middle-income customers. The platform aims to expand quick-gaining opportunities through trading marketplace for NFTs traders. A straightforward, transparent, and trustworthy service designed of the platform would guarantee a secured investment experience for any investor. Opportunities are made equal to all, even to the penny investment budget.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the METAIN REIT Smart Contracts.

It was conducted on commit `fe7669b82a7c3611daa97995a31cf3d88b1a6284` from git repository *https://github.com/metain-io/smc-reit-nft*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
48164f62a673884a54c285fb7b2423aa79d24419878c023d809abc1307935b18	./REITINO.sol
0c84980b192bacc7d5e4239a07054b71f62897f29a125bc734f89e38d6025fe2	./REITNFT.sol
ad0f6d5f0991621ca1b3690d8999c3fae061206cc71dbd98f12bca2c72a857ca	./shared/BlacklistingUpgradeable.sol
42ab9c9e08b4e8502b6a76c6c864fe368421172285f2587f258039a837eac4ff	./shared/ERC1155Tradable.sol
92a66224e0077dd6911e22b9e85461e0fdfcf0a679a2be26ce6d9ae0a9111370	./shared/ERC1155Upgradeable.sol
5ddf72723b7ed4da5b76e3a3e5fdde801eb5446c5eee9a360ac31fb539f27975	./shared/GovernableUpgradeable.sol
a226a016ca28412023395c7a04ae8477f62b2e6ed3ddecd8e72d4ba57eb3355e	./shared/IREITTradable.sol
c5c9d221306de33e031b38adaf0f1ae2146b35b1c3ed42fe2852ae3b7c1ffc67	./shared/KYCAccessUpgradeable.sol

96f67913bb8a24e7ff32b256145f5e03b002d7651f5741c05818228be1872ef7	./shared/LoyaltyProgram.sol
e8ee347a2615c9243f22f7c9f0c411ae5ebbeb0c20270a9580de30d756b80083	./shared/WhitelistingUpgradeable.sol

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.

SEVERITY LEVEL	DESCRIPTION
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The METAIN REIT Smart Contracts was written in `Solidity` language, with the required version to be `^0.8.9`. The source code was written based on OpenZeppelin's library.

METAIN REIT Smart Contracts contains 2 main contracts: `REITNFT` and `REITINO`.

2.1.1. REITNFT

The purpose of the `REITNFT` smart contract is to record the ownership of shareholders and to pay dividends and clearances in cryptocurrencies to them. There are some management roles in the contract:

- The governor can assign administrator roles for some parts of the smart contract. The primary role of the governor is to withdraw funds to trusted addresses and assign NFT creators.
- Administrator governs less critical parts of the smart contract like KYC, Loyalty Program and Whitelisting.
- REIT NFT creators who manage REIT Fund. There can be many creators that manage many funds.

REIT NFT shareholders (REIT NFT owners) can claim dividends whenever the creators fund and unlock the dividend vault. They can also forfeit all their NFT ownership to claim their clearance shares when the creators agree to release liquidation. REIT NFT transfer will be taxed to the receiver. Users can stake `_stakeableToken` tokens to gain a better loyalty level. A higher loyalty level means a higher purchase limit at INO and a lower transfer tax.

2.1.2. REITINO

The `REITINO` smart contract is used for the initial offering of REIT NFT. Users can buy REIT NFTs with payable tokens but only KYC users can claim REIT NFT.

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

METAIN fixed the code, according to Verichains's draft report, in commit `15f5c8b098eb7245c0e80e6522fef26c3660a2a3`.

2.2.1. Reentrancy with `ERC1155 safeTransferFrom` and `_mint` CRITICAL

Affected files:

- REITNFT.sol
- REITINO.sol

In `ERC1155`, when `safeTransferFrom` or `mint`, the contract call `_doSafeTransferAcceptanceCheck` to check if receiver is a contract then call `onERC1155Received` of that contract which may lead to reentrancy attack.

In `claimPendingBalances` function, attackers can use contract to claim pending balance and trigger reentrancy when `_nft.safeTransferFrom(address(this), account, id, quantity, empty)` to claim pending again and again because at this time, `_pendingBalances[id][account] = 0` has not been triggered, so they can drain all available balance of this contract.

In `purchaseWithToken` function, attackers can use contract with reentrancy to purchase more than `purchaseableAmount` because at transfer time, `_totalPurchasedByAccount[id][account] += quantity` has not been triggered.

In `createREIT` function, if `_initialOwner` is attacker's contract, he can trigger reentrancy when `_mint(_initialOwner, _id, _initialSupply, _data)` to mint any amount of tokens and at the end, `tokenSupply` of his `id` will be reset to `_initialSupply`.

```
function createREIT(
    address _initialOwner,
    uint256 _initialSupply,
    string calldata _uri,
    bytes calldata _data
) external onlyGovernor returns (uint256) {
    uint256 _id = super._getNextTokenID();
    super._incrementTokenTypeId();
    _setCreator(_initialOwner, _id);

    if (bytes(_uri).length > 0) {
        emit URI(_uri, _id);
        tokenUri[_id] = _uri;
    }

    tokenMetadata[_id] = TokenMetadata(0, new uint256[](1));
    tokenDividendData[_id] = TokenDividendData(
        0,
        0,
        new uint256[](MAX_DIVIDENDS_SEQUENCE_COUNT),
        0
    );

    _mint(_initialOwner, _id, _initialSupply, _data);

    tokenSupply[_id] = _initialSupply;

    emit Create(_id, _initialOwner, _initialSupply, _uri);
```

```
    return _id;
}

function purchaseWithToken(string calldata token, uint256 id, uint256 quantity)
    external
{
    require(_nft.isINOContract(address(this)), "IPO_ERR");

    uint256 stock = _nft.balanceOf(address(this), id) - _totalPendingAmount[id];
    require(stock >= quantity, "BALANCE_ERR");

    uint256 price = _nft.getINOUnitPrice(id);
    require(price > 0, "PRICE_ERR");

    address account = _msgSender();

    uint loyalty = _nft.getLoyaltyLevel(account);
    uint purchaseableAmount = getPurchasableLimit(id, loyalty);
    require(_totalPurchasedByAccount[id][account] + quantity <= purchaseableAmount,
"MAX_PURCHASE_REACHED");

    uint256 amount = price * quantity;
    require(
        _payableToken[token].transferFrom(
            account,
            address(this),
            amount
        ),
        "FUND_ERR"
    );

    if (_nft.isKYC(account)) {
        bytes memory empty;

        // IPO contract will register the balance without transfer fee
        _nft.safeTransferFrom(address(this), account, id, quantity, empty);
    } else {
        unchecked {
            // Pending this purchase until buyer is KYC
            _pendingBalances[id][account] += quantity;
            _totalPendingAmount[id] += quantity;
        }
    }

    _totalPurchasedByAccount[id][account] += quantity;
    _totalPurchased[id] += quantity;
}

function claimPendingBalances(uint256 id)
```

```
    external
{
    address account = _msgSender();

    require(_nft.isKYC(account), "KYC");
    require(_pendingBalances[id][account] > 0, "ZERO_BALANCE");

    uint256 quantity = _pendingBalances[id][account];
    uint256 stock = _nft.balanceOf(address(this), id);
    require(stock >= quantity, "BALANCE_ERR");

    bytes memory empty;
    _nft.safeTransferFrom(address(this), account, id, quantity, empty);

    unchecked {
        _pendingBalances[id][account] = 0;
        _totalPendingAmount[id] -= quantity;
    }
}

function _doSafeTransferAcceptanceCheck(
    address operator,
    address from,
    address to,
    uint256 id,
    uint256 amount,
    bytes memory data
) internal {
    if (to.isContract()) {
        try IERC1155ReceiverUpgradeable(to).onERC1155Received(operator, from, id, amount,
data) returns (bytes4 response) {
            if (response != IERC1155ReceiverUpgradeable.onERC1155Received.selector) {
                revert("REJECTED_TOKEN");
            }
        } catch Error(string memory reason) {
            revert(reason);
        } catch {
            revert("NON_IMPLEMENT");
        }
    }
}
```

RECOMMENDATION

Adding `nonReentrant` to ALL external/public functions and/or use Checks-Effects-Interactions pattern *https://docs.soliditylang.org/en/latest/security-considerations.html*.

```
function createREIT(
    address _initialOwner,
    uint256 _initialSupply,
```

```
    string calldata _uri,
    bytes calldata _data
) external onlyGovernor returns (uint256) {
    uint256 _id = super._getNextTokenID();
    super._incrementTokenTypeId();
    _setCreator(_initialOwner, _id);

    if (bytes(_uri).length > 0) {
        emit URI(_uri, _id);
        tokenUri[_id] = _uri;
    }

    tokenMetadata[_id] = TokenMetadata(0, new uint256[](1));
    tokenDividendData[_id] = TokenDividendData(
        0,
        0,
        new uint256[](MAX_DIVIDENDS_SEQUENCE_COUNT),
        0
    );

    tokenSupply[_id] = _initialSupply;

    _mint(_initialOwner, _id, _initialSupply, _data); // Interactions after all Effects

    emit Create(_id, _initialOwner, _initialSupply, _uri);
    return _id;
}

function purchaseWithToken(string calldata token, uint256 id, uint256 quantity)
    external
    nonReentrant
{
    require(_nft.isINOContract(address(this)), "IPO_ERR");

    uint256 stock = _nft.balanceOf(address(this), id) - _totalPendingAmount[id];
    require(stock >= quantity, "BALANCE_ERR");

    uint256 price = _nft.getINOUnitPrice(id);
    require(price > 0, "PRICE_ERR");

    address account = _msgSender();

    uint loyalty = _nft.getLoyaltyLevel(account);
    uint purchaseableAmount = getPurchasableLimit(id, loyalty);
    require(_totalPurchasedByAccount[id][account] + quantity <= purchaseableAmount,
"MAX_PURCHASE_REACHED");

    uint256 amount = price * quantity;
    require(
```

```
        _payableToken[token].transferFrom(
            account,
            address(this),
            amount
        ),
        "FUND_ERR"
    );

    _totalPurchasedByAccount[id][account] += quantity;
    _totalPurchased[id] += quantity;

    if (_nft.isKYC(account)) {
        bytes memory empty;

        // IPO contract will register the balance without transfer fee
        _nft.safeTransferFrom(address(this), account, id, quantity, empty); // Interactions
after all Effects
    } else {
        unchecked {
            // Pending this purchase until buyer is KYC
            _pendingBalances[id][account] += quantity;
            _totalPendingAmount[id] += quantity;
        }
    }
}

function claimPendingBalances(uint256 id)
    external
    nonReentrant
{
    address account = _msgSender();

    require(_nft.isKYC(account), "KYC");
    require(_pendingBalances[id][account] > 0, "ZERO_BALANCE");

    uint256 quantity = _pendingBalances[id][account];
    uint256 stock = _nft.balanceOf(address(this), id);
    require(stock >= quantity, "BALANCE_ERR");

    bytes memory empty;

    unchecked {
        _pendingBalances[id][account] = 0;
        _totalPendingAmount[id] -= quantity;
    }

    _nft.safeTransferFrom(address(this), account, id, quantity, empty); // Interactions
after all Effects
}
```

UPDATES

- *Oct 13, 2022*: This issue has been acknowledged and fixed by the METAIN team.

2.2.2. Unable to `removeCollaborator` HIGH

Affected files:

- REITNFT.sol

`_collaborators[account]` is required to be not `true` so we can not remove collaborator.

```
function _setCollaborator(address account) private {
    _collaborators[account] = true;
}

function removeCollaborator(address account) external onlyGovernor {
    require(_collaborators[account] != true, "NOT_A_COLLABORATOR"); // must be == true
    _removeCollaborator(account);
}

function _removeCollaborator(address account) private {
    _collaborators[account] = false;
}
```

RECOMMENDATION

Changing requirement for `_collaborators[account]` to be `true`.

```
function removeCollaborator(address account) external onlyGovernor {
    require(_collaborators[account], "NOT_A_COLLABORATOR");
    _removeCollaborator(account);
}
```

UPDATES

- *Oct 14, 2022*: This issue has been acknowledged and fixed by the METAIN team.

2.2.3. Same KYC accounts transfer can reset liability amount to zero HIGH

Affected files:

- REITNFT.sol

When `safeTransferFrom` from same KYC accounts, the `liabilityAmount` is transfer to new one, but it is replaced instead of increasing so the old account only need to do `safeTransferFrom` again with `amount` zero to reset liability amount of new account to zero. The same for `safeBatchTransferFrom`.

```
function safeTransferFrom(
    address from,
    address to,
    uint256 id,
    uint256 amount,
    bytes memory data
) public virtual override(ERC1155Upgradeable, IREITTradable) {
    ...
    // Same KYC case: transfer the liability without new one
    if (!isCollaboratorTransfer && isSameKYC) {
        tokenYieldVesting[id][to].liabilityAmount =
tokenYieldVesting[id][from].liabilityAmount;
        tokenYieldVesting[id][from].liabilityAmount = 0;
    }
    ...
}
```

RECOMMENDATION

Increasing liability amount of new account instead of replacing in `safeTransferFrom` and `safeBatchTransferFrom`.

```
function safeTransferFrom(
    address from,
    address to,
    uint256 id,
    uint256 amount,
    bytes memory data
) public virtual override(ERC1155Upgradeable, IREITTradable) {
    ...
    // Same KYC case: transfer the liability without new one
    if (!isCollaboratorTransfer && isSameKYC) {
    tokenYieldVesting[id][to].liabilityAmount +=
tokenYieldVesting[id][from].liabilityAmount; // increasing liability amount
    tokenYieldVesting[id][from].liabilityAmount = 0;
    }
...
}

    function safeBatchTransferFrom(
        address from,
        address to,
        uint256[] memory ids,
        uint256[] memory amounts,
        bytes memory data
    ) public virtual override(ERC1155Upgradeable, IREITTradable) {
        ...
        // Same KYC case: transfer the liability without new one
        if (!isCollaboratorTransfer && isSameKYC) {
        tokenYieldVesting[id][to].liabilityAmount +=
```

```
tokenYieldVesting[id][from].liabilityAmount; // increasing liability amount
        tokenYieldVesting[id][from].liabilityAmount = 0;
        }
    ...
    }
```

2.2.4. Cannot withdraw all loyalty staking tokens LOW

Affected files:

- REITNFT.sol

`Governor` cannot withdraw all loyalty staking tokens because of require statement `balance > amount`.

```
function withdrawLoyaltyStakings(uint256 amount) external onlyGovernor {
    require(amount > 0, "ZERO_AMOUNT");

    uint256 balance = _stakeableToken.balanceOf(address(this));
    require(balance > amount, "NOT_ENOUGH"); // should be >=

    _stakeableToken.transfer(governor(), amount);
}
```

RECOMMENDATION

Should require `balance >= amount`.

```
function withdrawLoyaltyStakings(uint256 amount) external onlyGovernor {
    require(amount > 0, "ZERO_AMOUNT");

    uint256 balance = _stakeableToken.balanceOf(address(this));
    require(balance >= amount, "NOT_ENOUGH"); // fix here
    _stakeableToken.transfer(governor(), amount);
}
```

UPDATES

- *Oct 13, 2022*: This issue has been acknowledged and fixed by the METAIN team.

2.2.5. Wrong `_liquidatedBalances` LOW

Affected files:

- REITNFT.sol

After an account `claimLiquidations`, if it receives more tokens and `claimLiquidations` again, `_liquidatedBalances` will be set to the amount of the later claim instead of all liquidated balances.

```
function claimLiquidations(uint256 id) external onlyKYC nonReentrant {
    ...
    unchecked {
        // Burn all shares belong to the sender
        _liquidatedBalances[id][account] = balance;
        _balances[id][account] = 0;

        liquidationVaultBalance[id] -= shareLiquidationValue;
        tokenYieldVesting[id][account]
        .claimedLiquidations += shareLiquidationValue;

        tokenYieldVesting[id][account].liabilityAmount = 0;
        tokenYieldVesting[id][account].taxPaid += taxAmount;
    }
}
```

RECOMMENDATION

Adding `balance` into `liquidatedBalances`.

```
function claimLiquidations(uint256 id) external onlyKYC nonReentrant {
    ...
    unchecked {
        // Burn all shares belong to the sender
        _liquidatedBalances[id][account] += balance; //fix here
        _balances[id][account] = 0;

        liquidationVaultBalance[id] -= shareLiquidationValue;
        tokenYieldVesting[id][account]
        .claimedLiquidations += shareLiquidationValue;

        tokenYieldVesting[id][account].liabilityAmount = 0;
        tokenYieldVesting[id][account].taxPaid += taxAmount;
    }
}
```

UPDATES

- *Oct 13, 2022*: This issue has been acknowledged and fixed by the METAIN team.

2.2.6. Upgradable contract without storage gaps LOW

Affected files:

- WhitelistingUpgradeable.sol
- KYCAccessUpgradeable.sol
- LoyaltyProgram.sol
- ERC1155Tradable.sol

`ERC1155Tradable`, `LoyaltyProgram`, `WhitelistingUpgradeable` and `KYCAccessUpgradeable` are base upgradable contracts, but they don't have storage gaps so in the future versions, those contracts cannot add more storage variables without affecting the storage layout of child contracts.

https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps

RECOMMENDATION

Consider adding storage gaps to base upgradable contracts.

UPDATES

- *Oct 13, 2022*: This issue has been resolved. The METAIN team only adds 1 gap storage because above contracts are almost completed and no need to add more features.

2.2.7. Wrong requirement in dev documentation INFORMATIVE

Affected files:

- REITNFT.sol

`settleLiableTransferFee` function require that only `shareholders` can execute but in the dev documentation, it requires owners in KYC list instead.

```
/**
 * @dev Pay the transfer fee now
 * Requirements:
 *
 * - Only owners in KYC list can execute
 *
 * @param id ID of the NFT
 */
function settleLiableTransferFee(uint256 id)
external
shareHoldersOnly(id)
```

RECOMMENDATION

Fixing the dev documentation.

```
/**
 * @dev Pay the transfer fee now
 * Requirements:
 *
 * - Only share holders can execute
 *
 * @param id ID of the NFT
 */
function settleLiableTransferFee(uint256 id)
```

```
external
shareHoldersOnly(id)
```

UPDATES

- *Oct 13, 2022*: This issue has been acknowledged and fixed by the METAIN team.

2.2.8. Unused native tokens INFORMATIVE

Affected files:

- REITINO.sol

`REITINO` contract enable receiving native tokens, but it does not use them. Consider removing `receive`, `fallback` and `withdrawFunds` functions.

```
receive() external payable {}

fallback() external payable {}
```

UPDATES

- *Oct 13, 2022*: This issue has been acknowledged and fixed by the METAIN team.

2.2.9. Unused variables INFORMATIVE

Affected files:

- REITNFT.sol

`beneficiary` of `YieldVesting` and `loyaltyRequirements` are unused variable. Consider removing them.

```
uint256[] private loyaltyRequirements;

struct YieldVesting {
    // beneficiary of the yield
    address beneficiary;
    // amount of dividends in pending
    uint256 pendingDividends;
    // total dividends claimed
    uint256 claimedDividends;
    // total liquidations claimed
    uint256 claimedLiquidations;
    // total tax paid
    uint256 taxPaid;
    // liability of this investor
    uint256 liabilityAmount;
    // can user claim liquidation?
    bool isLiquidationUnlocked;
```

```
    // intialization flag
    bool initialized;
    // index at dividend payments that was claimed
    uint32 lastClaimIndex;
}
```

UPDATES

- *Oct 13, 2022*: This issue has been acknowledged and fixed by the METAIN team.

2.2.10. Should require amount greater than 0 INFORMATIVE

Affected files:

- LoyaltyProgram.sol

`unstake` function should require `stakings[account].amount > 0` to avoid unnecessary zero transfer. The contract can also remove `safemath` because solidity `0.8.0+` already do that by default.

```
function unstake() external nonReentrant {1
    address account = _msgSender();
    uint256 elapsed = block.timestamp.sub(_stakings[account].startTime);
    require(elapsed > _minimumStakingPeriod, "TIME_ERR");

    require(_stakeableToken.transfer(account, _stakings[account].amount), "TRANFER_ERR");
    _stakings[account].amount = 0;
    _stakings[account].level = 0;
}
```

UPDATES

- *Oct 13, 2022*: This issue has been acknowledged and fixed by the METAIN team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 17, 2022*	Public Report	Verichains Lab

Table 2. Report versions history