# Smart Contract Security Audit Report

## MortgageFi

# 1. Contents

1. Contents ..... 2
2. General Information ..... 3
2.1. Introduction ..... 3
2.2. Scope of Work ..... 3
2.3. Threat Model ..... 3
2.4. Weakness Scoring ..... 4
2.5. Disclaimer ..... 4
3. Summary ..... 5
3.1. Suggestions ..... 5
4. General Recommendations ..... 6
4.1. Security Process Improvement ..... 6
5. Findings ..... 7
5.1. `fetch` is not invoked during `createContractByob` ..... 7
5.2. Lack of sequencer check ..... 7
5.3. Centralization risks ..... 8
5.4. `safeTransfer` is not used ..... 8
5.5. Chainlink stale data ..... 9
5.6. Stablecoin blacklisted users can't withdraw their money ..... 10
5.7. Hardcoded values ..... 10
6. Appendix ..... 12
6.1. About us ..... 12

# 2. General Information

This report contains information about the results of the security audit of the MortgageFi (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 05/03/2025 to 10/03/2025.

## 2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

## 2.2. Scope of Work

The audit scope included the contracts in the following pull request: MortgageFi/pull/3. Re-testing was done for the following pull request: MortgageFi/pull/5.

The following contracts have been tested:

- mortgagefipoolwbtcusdtupgraded.sol

## 2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User
- Protocol owner

## 2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

## 2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

# 3. Summary

During audit we have detected critical and high issues.

## 3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of March 31, 2025.

*Table. Discovered weaknesses*

| Issue | Contract | Risk Level | Status |
|---|---|---|---|
| `fetch` is not invoked during `createContractByob` | mortgagefipoolwbtcusdtupgraded.sol | **Critical** | Fixed |
| Lack of sequencer check | mortgagefipoolwbtcusdtupgraded.sol | **Medium** | Fixed |
| Centralization risks | mortgagefipoolwbtcusdtupgraded.sol | **Medium** | Acknowledged |
| `safeTransfer` is not used | mortgagefipoolwbtcusdtupgraded.sol | **Medium** | Fixed |
| Chainlink stale data | mortgagefipoolwbtcusdtupgraded.sol | **Medium** | Fixed |
| Stablecoin blacklisted users can't withdraw their money | mortgagefipoolwbtcusdtupgraded.sol | **Low** | Acknowledged |
| Hardcoded values | mortgagefipoolwbtcusdtupgraded.sol | **Info** | Acknowledged |

# 4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

## 4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

# 5. Findings

## 5.1. `fetch` is not invoked during `createContractByob`

**Risk Level**: Critical

**Status**: Fixed in the pull 5

**Contracts**:

- mortgagefipoolwbtcusdtupgraded.sol

**Location**: Function: `createContractByob`.

**Description:**

When a user invokes `createContractByob` they get shares for their contract coins. However, the `_mint` function only increases the user's balance without invoking `fetch`. This results in the fact that the user's balance `lastPoints` is 0 at the moment when he gets his shares. This leads to the following circumstances:

- User will have an insanely high yield because `Owing` assumes he was holding tokens from 0 timestamp, thus resulting in a drain of unclaimed balance by the user.
- In case unclaimed value is very low, it will be impossible to transfer tokens for any other users because of the underflow at `unclaimed = unclaimed - owing;`

fetch is also not invoked when minting shares to feeReceiver.

**Remediation:**

Consider invoking `fetch` during `createContractByob` execution and whenever shares to feeReceiver are minted.

## 5.2. Lack of sequencer check

**Risk Level**: Medium

**Status**: Fixed in the pull 4

**Contracts**:

- mortgagefipoolwbtcusdtupgraded.sol

**Description:**

When using Chainlink or other oracles with L2 chains like Arbitrum, smart contracts should check whether the L2 Sequencer is down to avoid stale pricing data that appears fresh.

**Remediation:**

Consider checking whether the L2 Sequencer is down

**References:**

- https://medium.com/Bima-Labs/chainlink-oracle-defi-attacks-93b6cb6541bf#0faf

## 5.3. Centralization risks

**Risk Level**: Medium

**Status**: Acknowledged

**Contracts**:

- mortgagefipoolwbtcusdtupgraded.sol

**Description:**

Admin can overwrite yield receiver for any user and take their rewards.

```
    // @audit stealing rewards
    function setOverwrite(address who, address receiver) external
onlyRole(ADMIN_ROLE) {
        // set address to overwrite the receiver of the yield (dexes,etc)
        yieldOverwrite[who] = receiver;
    }
```

**Remediation:**

Users should also have an ability to set an address of yield receiver.

## 5.4. `safeTransfer` is not used

**Risk Level**: Medium

**Status**: Fixed in the pull 4

**Contracts**:

- mortgagefipoolwbtcusdtupgraded.sol

**Description:**

Certain tokens may not adhere to the ERC20 standard completely, yet they are generally accepted by most code designed for ERC20 tokens. An instance of this is Tether (USDT), where the `transfer()` and `transferFrom()` functions on L1 do not conform to the specification's requirement of returning booleans; instead, they have no return value. Consequently, when such tokens are cast to IERC20, their function signatures do not align, leading to reverted calls.

```
104:          stablecoin.transfer(conversionVault, _tosend);

165:          stablecoin.transfer(msg.sender, size);

326:          contractCoin.transfer(msg.sender,coinSize[_nftID]);
```

**Remediation:**

Consider using the `safeTransfer()` function from the `SafeERC20` library instead of the `transfer()` function.

**References:**

- https://docs.openzeppelin.com/contracts/5.x/api/token/erc20#SafeERC20

## 5.5. Chainlink stale data

**Risk Level**: Medium

**Status**: Fixed in the pull 4

**Contracts**:

- mortgagefipoolwbtcusdtupgraded.sol

**Location**: Function: `createContractByob()`.

**Description:**

Inside `createContractByob` function price is retrieved from Chainlink's oracle service. However, there is a potential issue if the oracle returns stale or outdated data. Without proper validation of the data's freshness or a check for anomalies, the function could return inaccurate or misleading results.

```
(,int256 answer , ,,) = ICLoracle(CLoracle).latestRoundData();
require(answer > 0 ,"Oracle Fail" );
```

**Remediation:**

It is recommended to add a check that compares the updatedAt return timestamp from latestRoundData() with the current block timestamp and ensure that the priceFeed is being updated with the required frequency.

**References:**

- https://docs.chain.link/data-feeds#monitoring-data-feeds

## 5.6. Stablecoin blacklisted users can't withdraw their money

**Risk Level**: Low

**Status**: Acknowledged

**Contracts**:

- mortgagefipoolwbtcusdtupgraded.sol

**Location**: Function: `fromLineToWallet()`.

**Description:**

```
function fromLineToWallet(uint256 size)giveYield() external {
    // when there is availabillity exit with funds from the line
    mySizeWaitingForExit[msg.sender] -= size;
    inExitLine -= size;
    stablecoin.transfer(msg.sender, size);
  }
```

**Remediation:**

Consider adding receiver for exit.

## 5.7. Hardcoded values

**Risk Level**: Info

**Status**: Acknowledged

**Contracts**:

- mortgagefipoolwbtcusdtupgraded.sol

**Description:**

Some values such as oracle address, utilizations percentages and fee percentages are hardcoded in code.

```
CLoracle = address(0x6ce185860a4963106506C203335A2910413708e9);

if(utilisationAfterSize < 9500){
    //fees = size*stables* utilisationAfterSize/(freeCoins *95000);
    fees = size *
stables*(7*9500+11*utilisationAfterSize)/(9500*(freeCoins)*100);
}
if(utilisationAfterSize >= 9500){
    //fees = size * stables* utilisationAfterSize/(freeCoins *95000) + size *
stables*(utilisationAfterSize - 9500)/(freeCoins*500);
fees = size *
stables*(7*9500+11*utilisationAfterSize)/(9500*(freeCoins)*100)+size *
stables*(utilisationAfterSize-9500)*1000/(freeCoins*1000*610);
}

if(duration <= 24){ownCoins = size - size*4*duration/100;}
if(duration <= 12){ownCoins = size/2;}
if(ownCoins<2000){ownCoins = 2000;}

fees = size * stables*(18)/((10**8)*100);
```

**Remediation:**

Consider using variables instead of hardcoded values

# 6. Appendix

## 6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.