decurity

Smart Contract Security Audit Report

MortgageFi

1. Contents

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 08/12/2024 to 08/16/2024.

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 repository: https://github.com/0xDerivadev/MortgageFi (commit 57577707cd6f735bab664b64e5e569f66fee4bcc). Retesting was performed for commit 9f1c29.

The following contracts have been tested:

- mortgageconversionvault.sol
- mortgagefeetickets.sol
- mortgagecontracts.sol
- mortgagefipoolwethusdc.sol
- ciabv2erc20.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

As a result of this work, we have discovered medium exploitable security issues.

The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

The MortgageFi team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of August 16, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Possible DOS due to division by 0	mortgagefipoolwethusdc.sol	Medium	Acknowledged
Users can bypass protocol fees	mortgagefipoolwethusdc.sol	Medium	Fixed
The owner of the contract NFT isn't checked in the payDownContract() function	mortgagefipoolwethusdc.sol	Medium	Acknowledged
The fee ticket owner isn't checked	mortgageconversionvault.sol	Medium	Fixed
Loss of funds due to the uninitialized conversionVault	mortgagefipoolwethusdc.sol	Medium	Acknowledged
SafeERC20 library is imported but never used	mortgageconversionvault.sol, mortgagefipoolwethusdc.sol	Medium	Fixed
Lack of events	mortgagefipoolwethusdc.sol	Low	Acknowledged
Potential token sweep	mortgagefipoolwethusdc.sol	Info	Acknowledged
Debug Event	mortgagefipoolwethusdc.sol	Info	Fixed
Hardcoded values	mortgagefipoolwethusdc.sol	Info	Acknowledged
Lack of `giveYield` modifier in `earlyRepayContractInFull` function	mortgagefipoolwethusdc.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. Possible DOS due to division by 0

Risk Level: **Medium**

Status: Acknowledged

Contracts:

- mortgagefipoolwethusdc.sol

Location: Lines: `109`. Function: `giveYield`.

Description:

`giveYield` modifier uses the following formula to update points:

`totalPoints = totalPoints + (_size * pointMultiplier / totalSupply);`

However, in case `totalSupply` is 0, this will cause a revert, because of division by 0.

It's possible to bring `totalSupply` to via the following method:

- Deposit 1000 stable coins in pool, now `totalSupply` is 1000.
- Invoke `createContract` for some amount
- Withdraw 1000 stable coins from pool, `totalSupply` is now 0, but buffer is also 0, so division by 0 is not happening.
- Wait till payment accrues and invoke `payDownContract` with very small amount, e.g 5 wei, so `fees/10` rounds down to 0 what result in 0 `_protocolTake`. However, buffer is not increased by a few wei of fees(depends on the fee size of the contract)
- Now, when buffer is non 0, but `totalSupply` is 0, users can't invoke any functions, because `giveYield` will revert due to division by 0.

Remediation:

Consider checking that `totalSupply` > 0 before diving.

5.2. Users can bypass protocol fees

Risk Level: **Medium**

Status: Fixed in commit 88df85

Contracts:

- mortgagefipoolwethusdc.sol

Location: Function: createContract.

Description:

mortgagefipool contract relies on free stables balance when calculating baseSize and feeSize of a position. Stables balance is calculated th following way: uint256 stables = totalSupply + unclaimed + buffer + inExitLine - stablesInContracts - stablecoin.balanceOf(address(this));

After that fees are calculated by the following way: if(utilisationAfterSize >= 9500){ fees = size * stables * utilisationAfterSize/(freeCoins*95000) + size * stables*(utilisationAfterSize - 9500)/(freeCoins*500); } fees = fees*duration; (There is another option, but we will stick to high utilization rate for our example).

And baseSize is calculated the following way: baseSize[_nftID] = stables* size/freeCoins ; As we can see both fees and baseSize are a product of stables and some other variables. This leads to a conclusion that if stables equals 0, both baseSize and fees will be 0. This will allow user to immediately invoke finishContract function and get his contract coins. Let's assume we have the following state before we invoke createContract:

contractCoin.balanceOf(address(this)) = 1050

coinsInContracts = 50

freeCoins = 1000

stables = 1000(we assume that after all calculations there are exactly 1000 stables)

User wants to borrow 994 contract coins for 30 years. In this case he will need to send 994/50 ownCoins(19.88) and coinsInContracts will become 69.88. At the same time contractCoin.balanceOf(address(this)) will become 1069.88 So, utilization rate will be (994 + 69.88) * 10000 / 1069.88 = 9943.918944180656

9943.918944180656 < 9950 so, utilization check passes.

fees per year will be: (994 * 1000 * 9943.918944180656) / (1000 * 95000) + (994 * 1000 * (9943.918944180656 - 9500)) / (1000 * 500) = 986.5556550365706

Total fees will be 986.5556550365706 * 30 = 29596.669651097116.

baseSize[_nftID] will be 1000 * 994 / 1000 = 994.

After that user can invoke earlyRepayContractInFull to immediately close his contract. This function takes earlyFeesfrom user. earlyFees are calculated the following way:

uint256 earlyFees = (baseSize[_nftID] - (amountPaid[_nftID] * baseSize[_nftID] /(baseSize[_nftID] + feeSize[_nftID])))/100; Let's assume that user has not paid anything for his contract yet so earlyFees will be basically baseSize[_nftID]/100.

This will result in earlyFees = 994 / 100 = 9.94 In total user will need to pay earlyFees + baseSize[_nftID] what will be 9.94 + 994 = 1003.94. Protocol will earn 9.94 / 10 in fees and protocol depositors will earn everything else from the fees.

However, as already said baseSize and fees calculation relies on stables. This allows user to donate stablecoins directly to contract address and bypass fees with the following exploit.

Let's assume we have the same state of the protocol as mentioned earlier. But before opening contract user sends 1000 of stablecoins directly to contract. This will result in stables to equal 0, because stables calculation relies on stablecoin.balanceOf(address(this));. So, if stablecoin balance increases stables value decreases.

With this manipulation both fees and baseSize[_nftID] will equal 0, because stables is now 0.

With this user can directly invoke finishContract function and amountPaid check will path because both feeSize[_nftID] and baseSize[_nftID] are 0. In this scenario user will have to pay 1000 stablecoins to get 994 contract coins, that saves him 3.94 stable coins for the same amount of contract coins he got via the previous method. Also, no fees will be accounted in buffer or protocol yield.

Note:

giveYield may be invoked in the same block before the manipulation(e.g. via deposit of 1 wei of stable coin, so it doesn't accidentally send stablecoins to conversion pool).

This technique may be also combined with a flash loan if worth to do so.

All numbers are simplified for calculations, we know about if(ownCoins<1 ether/10000){ownCoins = 1 ether/10000;} check, decimals were removed, because not important in this case.

Remediation:

Consider adding new variable to store balance of stable coins instead of using `stablecoin.balanceOf(address(this));`

5.3. The owner of the contract NFT isn't checked in the payDownContract() function

Risk Level: Medium

Status: Client: Intended behavior

Contracts:

- mortgagefipoolwethusdc.sol

Location: Lines: `202`. Function: `payDownContract()`.

Description:

The function `payDownContract()` of the `mortgagefipool` contract hasn't a check for the owner of the loan contract NFT. Despite the possibility of paying for other contracts, if the NFT is burned, the payment will still go through, resulting in the loss of funds. This occurs because it's not possible to finish the contract after that once the NFT has been burned.

Remediation:

Add the check for the `msg.sender` NFT ownership to the `payDownContract()` function in the same way using the ERC721 `ownerOf()` function in the same way as it is implemented in functions `earlyRepayContractInFull()` and `finishContract()`. The ERC721 `ownerOf` function assumes that NFTs assigned to zero address are considered invalid, and queries about them revert with the corresponding error. Such ownership mitigates both payments for someone else's contract and accidental payments for contracts with burned NFTs.

5.4. The fee ticket owner isn't checked

Risk Level: Medium

Status: Fixed in commit 1839e4

Contracts:

- mortgageconversionvault.sol

Location: Lines: 217.

Description:

In the mortgageconversionvault contract, during the entryCoin deposit or minting process, a fee ticket is minted by the vault through the afterDeposit() function. Later, when withdraw() or redeem() is called, this ticket is used within the beforeWithdraw() function. The vault uses the ticket by calling the use() function, passing the corresponding _nftID provided by the user. However, since the ownership of the NFT is not verified during the withdrawal phase, an attacker uses another user's _nftID to withdraw their own funds.

Each ticket has a delay of 3 minutes before funds can be withdrawn after the entryCoin deposit. This delay can be bypassed by using another user's NFT. Moreover, the affected user may be unable to fully withdraw their own funds later, as their NFT might have already been partially or fully used. This creates a DoS opportunity, where the attacker deposits own funds and utilizes other users' tickets for withdrawals, preventing users to withdraw their funds. In such a case it requires to mint tickets to users manually by the address with the MINTER_ROLE on the mortgagefeetickets contract to return the funds.

Remediation:

Add a check to the beforeWithdraw() function in the mortgageconversionvault contract to verify that the msg.sender is the owner of the fee ticket with the provided _nftID .

5.5. Loss of funds due to the uninitialized conversionVault

Risk Level: **Medium**

Status: Acknowledged

Contracts:

- mortgagefipoolwethusdc.sol

Location: Lines: 94. Function: giveYield().

Description:

It's possible to have a pool with an uninitialized conversionVault address because its update is separated from the pool's constructor() function. In the worst-case scenario (e.g., on the mainnet with

USDT stablecoin), yield from deposits could be burned during the execution of the `giveYield()` modifier, as the yield would be transferred to the zero address. Since USDT allows transfers to zero addresses, this operation would not trigger a revert.

```
File: mortgagefipoolwethusdc.sol
    modifier giveYield() {
        // lazy send to conversion to give predictable flow
        uint256 myBalance = stablecoin.balanceOf(address(this));
        if(myBalance > inExitLine){
            // do it
            uint256 _delta = myBalance - inExitLine;
            uint256 _tosend = _delta;
            if(block.timestamp - lastYieldInteraction <= 3 days){
                _tosend = _delta* (block.timestamp - lastYieldInteraction)/3 days;
            }
            stablecoin.transfer(conversionVault, _tosend);
        }
        // yield it
        if(buffer > 0){
            // calculate size
            uint256 _size;
            if(block.timestamp - lastYieldInteraction > 30 days){
                _size = buffer;
                buffer = 0;
            }
            if(block.timestamp - lastYieldInteraction <= 30 days){
                _size = buffer* (block.timestamp - lastYieldInteraction)/30 days;
                buffer -= _size;
            }

        totalPoints = totalPoints + (_size * pointMultiplier / totalSupply);
        unclaimed += _size;
        }
        lastYieldInteraction = block.timestamp;
        _;
    }
```

Same problem has `feeReceiver`. Part of the fees can be minted to a zero address.

Remediation:

Consider checking `conversionVault` address is not equal to 0x0 before the yield transfer, checking `feeReceiver` address is not equal to 0x0 before fee transfer.

5.6. SafeERC20 library is imported but never used

Risk Level: Medium

Status: Fixed in commit 9f1c29

Contracts:

- mortgageconversionvault.sol,
- mortgagefipoolwethusdc.sol

Description:

The contracts `MortgageConversionVault` and `MortgageFiPool` have imported `SafeERC20`, but they are still using the default `ERC20` methods `transfer` and `transferFrom` instead of safe versions. Some tokens, such as USDT on the mainnet, can return void on these calls, which can lead to a revert due to interface inconsistencies.

Remediation:

Consider using a safe version of the mentioned functions from SafeERC20 library, such as `safeTransfer()`, `safeTransferFrom()` and `forceApprove()` , that handles a case when functions may not return values.

References:

- https://github.com/d-xo/weird-erc20#missing-return-values

5.7. Lack of events

Risk Level: Low

Status: Acknowledged

Contracts:

- mortgagefipoolwethusdc.sol

Description:

The following functions perform state changes and are important parts of logic, however, they don't emit an event:

```
deposit()
burnToGetOutInLine()
fromLineToWallet()
createContract()
payDownContract()
finishContract()
earlyRepayContractInFull()
defaultContract()
setFeeReceiver()
setConversionVault()
```

Remediation:

Consider adding events to the listed functions.

5.8. Potential token sweep

Risk Level: Info

Status: Acknowledged

Contracts:

- mortgagefipoolwethusdc.sol

Location: Function: `createContract`.

Description:

In case there are `entryCoins` in `mortgagefipool` and no one has yet deposited stable coins to it, users can basically take them for free, because `baseSize` and `feeSize` will be 0.

Remediation:

Consider carefully deploying contracts, so `entryCoins` won't accidentally go to pool before any deposit. Also, we would recommend to check that `stables` ≠ 0 in `createContract` as a sanity check.

5.9. Debug Event

Risk Level: Info

Status: Fixed in 88df85

Contracts:

- mortgagefipoolwethusdc.sol

Location: Lines: 377.

Description:

Contract `mortgagefipool` contain a debug event:

```
File: mortgagefipoolwethusdc.sol
377:     event _debug(uint256 indexed amount);
```

Remediation:

Consider removing all debug events from production code.

5.10. Hardcoded values

Risk Level: **Info**

Status: Acknowledged

Contracts:

- mortgagefipoolwethusdc.sol

Description:

Contract `mortgagefipool` has a lot of hardcoded values:

```
File: mortgagefipoolwethusdc.sol
166:         uint256 ownCoins =  size/50;
167:         if(duration <= 24){ownCoins = size - size*4*duration/100;}
168:         if(duration <= 12){ownCoins = size/2;}
169:         if(ownCoins<1 ether/10000){ownCoins = 1 ether/10000;}
175:         uint256 utilisationAfterSize = (size +
coinsInContracts)*10000/contractCoin.balanceOf(address(this));//in 10000
points
176:         require(utilisationAfterSize <= 9950,"utilisation protection <=
99.5%");
178:         if(utilisationAfterSize < 9500){
179:             fees = size * stables *
utilisationAfterSize/(freeCoins*95000);
180:         }
181:         if(utilisationAfterSize >= 9500){
182:             fees = size * stables *
utilisationAfterSize/(freeCoins*95000) + size * stables *
(utilisationAfterSize - 9500)/(freeCoins*500);
183:         }
200:         IERC20Extended(rewardsCoin).mint(ref,stables*1000000000000*
size/freeCoins);
```

```
          secondsTillLiq = 3888000*(_defaultAtThisSize -
_currentPaymentPending)/_defaultAtThisSize;
```

It increases the risks of bugs due to the developer mistakes in the future.

Remediation:

Consider using variables instead of hardcoded values.

5.11. Lack of `giveYield` modifier in `earlyRepayContractInFull` function

Risk Level: **Info**

Status: Acknowledged

Contracts:

- mortgagefipoolwethusdc.sol

Location: Lines: 233. Function: `earlyRepayContractInFull`.

Description:

Every state-changing function in pool has a modifier `giveYield()`, except the `earlyRepayContractInFull()`.

Remediation:

Consider to add `giveYield()` modifier to the `earlyRepayContractInFull()` function for code consistency.

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.