CLEARPOOL SECURITY AUDIT REPORT

March 31, 2023

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals

- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Clearpool is a decentralized finance platform that offers both permissionless and permissioned liquidity pools for institutional borrowers and lenders. The permissionless pools allow anyone to lend directly to whitelisted institutions and earn risk-adjusted returns, while the permissioned pools provide a fully compliant marketplace for wholesale borrowing and lending of digital assets.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Clearpool
Project name	Auction
Timeline	19.12.2022 - 31.03.2023
Number of Auditors	4

Project Log

Date	Commit Hash	Note
13.12.2022	f1f38c5ba83e2c3a68155042229b04678d9425ce	Commit for the audit

Date	Commit Hash	Note
18.01.2023	68820cb61c8e3fa1714be0aca161d9916c8ac9d3	Commit for the re-audit
30.01.2023	2d36634e87dbcb494120f7f1ef09ace9ac2fab90	Commit for the re-audit v2
03.03.2023	70984609d86f4dda32a0eae56dc435c6b1527d5f	Commit for the re-audit v3

Project Scope

The audit covered the following files:

File name	Link
contracts/PoolFactory.sol	PoolFactory.sol
contracts/Auction.sol	Auction.sol
contracts/libraries/Decimal.sol	Decimal.sol

Deployments

Network: Ethereum

Contract	Address	Creation TX hash
PoolFactory	0x99C10A7aBd93b2db6d1a2271e69F268a2c356b80	0x0f324fae52add92950618855bec3207fa40cd093090018e18863cc5536763840
Auction	0xCE3Fec90A05992dF1357651FEF6D143FeeC7Ca16	0xca670a5b629df45497dbea9d6e909842f9fb53964b2a3b9d8cba9a78046e1391

Network: Polygon

Contract	Address	Creation TX hash
PoolFactory	0x3B0bbd7E5877d64aD5886dDe14ca5cEecC29D55B	0xc67cec6210c03c411caf84c3c50f3bdb3ccc76952fc726f8213e74277d9c6140
Auction	0xF1F6626BC305331261B755225ecDfc993500fE31	0x4e0edcf0de13bbc1832d2d2e3b33cdf9b8776f136ad47d79bf576112447f6b3e

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	6
Medium	3
Low	12

ID	Name	Severity	Status
H-1	Front-running for the `bid()` method	High	Fixed
H-2	Rejected auction is always in default state	High	Fixed
H-3	Protect CPOOLs in PoolFactory	High	Fixed
H-4	Infinite Block Auction	High	Fixed
H-5	Double-Adding Pool Vulnerability	High	Fixed
H-6	Close active pools via `resolveAuctionWithoutGoverment`	High	Fixed
M-1	Risks of non-complete initialization	Medium	Fixed
M-2	Stake is locked for the contract owner in `_createPool()`	Medium	Acknowledged
M-3	The pool borrower must not be permitted to participate in the auction	Medium	Fixed
L-1	Unused `flashGovernor` variable	Low	Fixed

L-2	PoolFactory functions `setReserveFactor()` and `setInsuranceFactor()` require additional checks	Low	Fixed
L-3	`warningUtilization` can be above `provisionalDefaultUtilization`	Low	Fixed
L-4	Add an auction resolution event	Low	Fixed
L-5	Parameters not validated for zero address	Low	Fixed
L-6	Excessive Centralization	Low	Acknowledged
L-7	Possibly unnecessary state change in `processDebtClaim()`	Low	Acknowledged
L-8	Reentrancy Vulnerability in `bid()`	Low	Fixed
L-9	Out-of-gas vulnerability in `closePool()`	Low	Acknowledged
L-10	`PoolFactory.closePool` doesn't reset `stakedAmount`	Low	Fixed
L-11	Parameters are not validated in `setPoolAuctionEnd`	Low	Fixed
L-12	The `amount` parameter is not validated in `increaseBid(address pool, uint256 amount)`	Low	Fixed

1.6 Conclusion

During the audit process, the developers spotted and acknowledged 6 HIGH, 3 MEDIUM, and 12 LOW severity findings. After working on the reported findings, all of them were acknowledged or fixed by the client.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	Front-running for the `bid()` method
Severity	High
Status	Fixed in 70984609

Description

The `bid()` transaction (Auction.sol#L105) with a slightly higher bid can be sent right before the end of an auction.

The winner can be the one who bids a unit more at the last moment. Every bid transaction can be front-run or back-run. Thus, a legitimate bidder can be prevented from making a proper bid.

Recommendation

We recommended adding a specific step of an auction and taking a bid in `resolveAuction` from the previous block to avoid front-running.

Client's commentary

> MixBytes: Major logic changes in commit `68820cb61c8e3fa1714be0aca161d9916c8ac9d3` that seriously complicates the project's logic without fixing the bug described in the finding.
> MixBytes: The problem was fixed in commit 70984609d86f4dda32a0eae56dc435c6b1527d5f. The main problem of this finding was that bidders may lose interest in depositing money, as there is a risk of depositing their funds for a long time and not winning at the last moment due to front-running.

H-2	Rejected auction is always in default state
Severity	High
Status	Fixed in 68820cb6

Description

After the auction has been rejected, there is no way for the owner or user to close the pool. So, in `PoolFactory.marketPools`, `PoolFactory.pools` this pool (with default status and rejected auction) will be there for infinity.

```
// start some auction
...
await auction.resolveAuction(pool.address, false);

await increaseTime(MUCH_TIME);

// try to close pool after Auction (calls from owner)
// await pool.allowWithdrawalAfterNoAuction(); — revert
// await pool.close(); — revert
```

Recommendation

We recommend adding the description of this state to the whitepaper or adding a conditional to close the pool:

- PoolBase.sol#L180.

Client's commentary

Fixed

H-3	Protect CPOOLs in PoolFactory
Severity	High
Status	Fixed in 68820cb6

Description

The rewards, which will be paid to the user, are at the factory.

`Sweep` allows to withdraw CPOOL tokens and block the possibility of claiming by users.

- PoolFactory.sol#L393

Recommendation

We recommended protecting CPOOL to avoid sweeping.

Client's commentary

Fixed

H-4	Infinite Block Auction
Severity	High
Status	Fixed in 68820cb6

Description

The owner may not call `resolveAuction` (Auction.sol#L139):

- to block `lastBid` of a user on the `Auction` contract and there is no mechanism to bypass that;
- to prevent the pool's users from getting an insurance or bid after the auction;
- to manipulate the market.

Recommendation

We recommended allowing users to call `resolveAuction` if enough time has passed.

Client's commentary

Fixed

H-5	Double-Adding Pool Vulnerability
Severity	High
Status	Fixed in 68820cb6

Description

The PoolFactory.sol#L214 function may add a pool to the `marketPools` array more than once if the function is called multiple times. This can occur if the function is called by mistake.

The PoolFactory.sol#L402 function is unable to remove more than one instance of the same pool from the `marketPools` array and cannot be called twice for the same pool. As a result, the pool will remain in the array misleading users and also consuming gas when the array is iterated through.

Recommendation

In order to fix this vulnerability, the `initializeExistingPoolsByMarket()` function should include an assertion that checks for the existence of the pool in the marketPools array before adding it. This will ensure that the same pool is not added more than once.

Client's commentary

Fixed

H-6	Close active pools via `resolveAuctionWithoutGoverment`
Severity	High
Status	Fixed in 2d36634e

Description

In commit 68820cb61c8e3fa1714be0aca161d9916c8ac9d3 `resolveAuctionWithoutGoverment` is called without a pool and time filtering. These checks can be passed by the user (Auction.sol#L266):

```
# by default currentAuction.tokenId = 0
require(currentAuction.tokenId == 0, "AC");
# by default currentAuction.end = 0
require(block.timestamp >= currentAuction.end, "ANF");
# block.timestamp >= 6 days
require(block.timestamp - currentAuction.end >= 6 days, "TNP");

# lastBidder = 0x0, lastBid = 0
# some erc-20 tokens allow to `transfer(0x0, 0)`
currency.safeTransfer(
    currentAuction.lastBidder,
    currentAuction.lastBid
);
```

So, anyone can close an active pool at any time and call (Auction.sol#L288):

```
...
auctionInfo[pool].tokenId = type(uint96).max;
...
IPoolMaster(pool).processDebtClaim();
...
```

Recommendation

We recommend checking the `currentAuction.end` and `pool` variables.

There is code that can block this flow (Auction.sol#L280):

```
currency.safeTransfer(
    currentAuction.lastBidder,
    currentAuction.lastBid
);
```

But WETH, USDT, etc. can allow to call `transfer(0x0, 0)`.

Client's commentary

Fixed

2.3 Medium

M-1	Risks of non-complete initialization
Severity	Medium
Status	Fixed in 68820cb6

Description

`PoolFactory.initialize()` is far from being the final initialization.
It sets only:

- `cpool`
- `staking`
- `poolBeacon`
- `interestRateModel`
- `auction`

But many of crucial parameters remain not set and require calling a separate function for every parameter.

- `treasury` - if not set, a reserve token can be transferred to a zero address on pool.proceesAuctionStart(), pool.close()
- `reserveFactor` - if not set, a pool will work wrong
- `insuranceFactor` - if not set, a pool will work wrong
- `warningUtilization` - if not set, a pool will work wrong
- `provisionalDefaultUtilization` - if not set, a pool will work wrong
- `warningGracePeriod` - if not set, a pool will work wrong
- `maxInactivePeriod` - if not set, a pool will work wrong
- `periodToStartAuction` - if not set, a pool will work wrong
- `rewardPerSecond` - if not set, cPool token rewards will be set zero, that is ok

It is not so bad that these parameters are not set during the factory initialization. The problem is that Factory allows createPool() if some of these parameters are not set.

- PoolFactory.sol#L181-201

This can lead to a situation where a human error causes one of these variables to be undefined, resulting in a new pool being created with incorrect parameters in the `PoolFactory` contract.

Recommendation

We recommend either putting all parameter settings to the initialization function or placing `require(parameter!=0)` for all parameters in _createPool().

Client's commentary

> Client's commentary: Fixed in commit 68820cb6.
> MixBytes: pool creation now checks setters except: `rewardPerSecond` - 0 is by defaul now.

M-2	Stake is locked for the contract owner in `_createPool()`
Severity	Medium
Status	Acknowledged

Description

The `_createPool()` method is called from the `createPoolInitial()` and `createPool()` methods which are marked with the `onlyOwner` modifier.
Thus,

```
info.staker = msg.sender;
info.stakedAmount = staking.lockStake(msg.sender);
```

at PoolFactory.sol#L509 uses the owner's address to lock the stake, while it's a manager who is supposed to be staking tokens according to the whitepaper:

> 2.1 Borrower Whitelisting & Credit Evaluation
> <...>
> To become whitelisted, the institution must first complete a due diligence, KYC and AML procedure, agree to the Clearpool Terms & Conditions, and stake an amount of CPOOL tokens before a pool can be launched.

Recommendation

If it is not an expected behavior, correct the lines so that the funds were taken from the right staker and this right staker were stored in `info.staker`.

Client's commentary

> Client's commentary: Decided not to be fixed because: Governor (owner) is the gatekeeper of who can open the pool, we help this user to stake tokens on their behalf to simplify their UX.

M-3	The pool borrower must not be permitted to participate in the auction
Severity	Medium
Status	Fixed in 68820cb6

Description

In `setWhitelistedBidder` there aren't any checks for `bidder` (Auction.sol#L179).

But in whitepaper:

> 3.3 Auction
> <…>
> Bidders can be individuals or institutions but must be whitelisted. Whitelisting is achieved through providing KYC information and by declaring the UBO of the bid. The pool borrower is not permitted to participate in the auction.

It requires excluding the manager of the pool.

Recommendation

Although a manager may create a different address, we recommended adding this check to `setWhitelistedBidder`:

- Auction.sol#L179

Or revising the whitepaper to make this logic more transparent.

Client's commentary

Fixed

2.4 Low

L-1	Unused `flashGovernor` variable
Severity	Low
Status	Fixed in 68820cb6

Description

The PoolFactory.sol#L26 variable is not being used in the code and is always zero.

Recommendation

Remove the unused field or mark it as obsolete if it's required for maintaining the storage structure.

Client's commentary

Fixed

L-2	PoolFactory functions `setReserveFactor()` and `setInsuranceFactor()` require additional checks
Severity	Low
Status	Fixed in 68820cb6

Description

Two versions of these functions found:

1. in PoolFacrory

```
function setReserveFactor(uint256 reserveFactor_) external onlyOwner {
        require(reserveFactor_ <= Decimal.ONE, "GTO");
        reserveFactor = reserveFactor_;
        emit ReserveFactorSet(reserveFactor_);
    }

function setInsuranceFactor(uint256 insuranceFactor_) external onlyOwner {
        require(insuranceFactor_ <= Decimal.ONE, "GTO");
        insuranceFactor = insuranceFactor_;
        emit InsuranceFactorSet(insuranceFactor_);
    }
```

2. in pools (PoolConfuguration.sol)

```
function setReserveFactor(uint256 reserveFactor_) external onlyGovernor {
        require(reserveFactor + insuranceFactor <= Decimal.ONE, "GTO");
        reserveFactor = reserveFactor_;
    }

function setInsuranceFactor(uint256 insuranceFactor_)
        external
        onlyGovernor
    {
        require(reserveFactor + insuranceFactor <= Decimal.ONE, "GTO");
        insuranceFactor = insuranceFactor_;
    }
```

The version in the pools checks that the sum of two factors is below Decimal.ONE. The factory setters do not check this, so pools can be created where the sum is above Decimal.ONE.

- PoolFactory.sol#L313-325

Recommendation

We recommend that the two functions should be synchronized and adding the following line to the Factory functions.

```
require(reserveFactor + insuranceFactor <= Decimal.ONE, "GTO");
```

Client's commentary

Fixed

L-3	`warningUtilization` can be above `provisionalDefaultUtilization`
Severity	Low
Status	Fixed in 2d36634e

Description

`WarningUtilization` is expected to be always below `ProvisionalDefaultUtilization` but setters do not check this.

```
function setWarningUtilization(uint256 warningUtilization_)
        external
        onlyOwner
    {
        require(warningUtilization_ <= Decimal.ONE, "GTO");
        warningUtilization = warningUtilization_;
        emit WarningUtilizationSet(warningUtilization_);
    }

function setProvisionalDefaultUtilization(
        uint256 provisionalDefaultUtilization_
    ) external onlyOwner {
        require(provisionalDefaultUtilization_ <= Decimal.ONE, "GTO");
        provisionalDefaultUtilization = provisionalDefaultUtilization_;
        emit ProvisionalDefaultUtilizationSet(
            provisionalDefaultUtilization_
        );
    }
```

- PoolFactory.sol#L329-346

Recommendation

We recommend adding the necessary comparison invariants in setters.

Client's commentary

> Client's commentary: Fixed in commit 2d36634e.

L-4	Add an auction resolution event
Severity	Low
Status	Fixed in 68820cb6

Description

The`resolveAuction()` method at Auction.sol#L139 doesn't emit an event about the auction resolution result.

Recommendation

Add an event to notify the interested parties about the auction resolution result.

Client's commentary

> Fixed

L-5	Parameters not validated for zero address
Severity	Low
Status	Fixed in 68820cb6

Description

The methods:

- `createPoolInitial()` at PoolFactory.sol#L225,
- `createPool()` at PoolFactory.sol#L238,
- `transferPool()` at PoolFactory.sol#L246,
- `setManagerInfo()` at PoolFactory.sol#L268,
- `setCurrency()` at PoolFactory.sol#L279

do not validate input for zero address, which allows to add not valid data to the storage.

Recommendation

Add a zero address validation.

Client's commentary

> Fixed

L-6	Excessive Centralization
Severity	Low
Status	Acknowledged

Description

In the `Auction` and `PoolFactory` contracts, all privileged methods are called by the contract owner.

- PoolFactory.sol
- Auction.sol

`setPoolAuctionEnd()` can be used by the owner to endlessly extend the auction or to finish it at any moment, for instance, after a certain bid.

Recommendation

In order to reduce centralization, it may be beneficial to use the OpenZeppelin's `AccessControl` contract to granulate roles. This will allow different roles to be defined and different levels of access to the contract's privileged methods.

This will enhance the security and decentralization of the contract by allowing multiple parties to have different levels of access and control. It will also make the contract more flexible and adaptable, as different roles can be created and modified as needed.

Client's commentary

> Decided not to fix because: AccessControl is managed with a multisig wallet.

L-7	Possibly unnecessary state change in `processDebtClaim()`
Severity	Low
Status	Acknowledged

Description

The `PoolBase.processDebtClaim()` function, which is called when an auction completes, has a line which sets the pool's state to `State.Default` (PoolBase.sol#L238):

```
_info.state = State.Default;
```

However, if the pool were in a different state, we would not be able to start an auction in the first place, meaning that this function would not be called. Therefore, this line is unnecessary.

Recommendation

To fix this issue, it is recommended to evaluate the conditions under which the `PoolBase.processDebtClaim()` function is called and determine whether the state change to `State.Default` is necessary. If it is not needed, the line of code should be removed to ensure that the state of the pool is not unnecessarily changed and that the function operates correctly.

Client's commentary

> The pool `Default` state is a virtual state (it allows the calculations to go through) before the Auction ends and a hard state after the Auction is resolved (blocks any further calculations).

L-8	Reentrancy Vulnerability in `bid()`
Severity	Low
Status	Fixed in 68820cb6

Description

An attacker who is able to bypass the whitelist and register a pool with an ERC-777 token or a poisoned custom token may be able to exploit the reentrancy vulnerability in the `Auction.bid()` function. This would allow them to drain more funds to their benefit than intended by the contract's logic.

The attacker could potentially bypass the whitelist through social engineering or other means.

- Auction.sol#L121

Recommendation

In order to fix this vulnerability, it is recommended to use the `nonReentrant` modifier from the OpenZeppelin's `ReentrancyGuard` contract in the `Auction.bid()` function. It is also recommended to add the `nonReentrant` modifier to the `Auction.resolveAuction()` function, just to be on the safe side. This will prevent these functions from being called recursively and will protect against reentrancy attacks.

Client's commentary

Fixed

L-9	Out-of-gas vulnerability in `closePool()`
Severity	Low
Status	Acknowledged

Description

The PoolFactory.sol#L402 function has a loop over both `pools.length` and `marketPools[market].length`. If these arrays become large enough, the function will stop working and will always return an out-of-gas error.

Recommendation

There are different approaches to solve the problem. One option is to restrict the maximum number of pools that can be added. Another option is to use only mappings instead of arrays.

Client's commentary

> Acknowledged

L-10	`PoolFactory.closePool` doesn't reset `stakedAmount`
Severity	Low
Status	Fixed in 68820cb6

Description

In `closePool`:

- PoolFactory.sol#L427

after unstaking `stakedAmount` won't be reset.

Recommendation

Although `closePool` calls once, we recommended setting `stakedAmount` as zero:

```
info.staker = address(0);
info.stakedAmount = 0;
```

Client's commentary

> Fixed

L-11	Parameters are not validated in `setPoolAuctionEnd`
Severity	Low
Status	Fixed in 70984609

Description

`setPoolAuctionEnd()` can be used by the owner to endlessly extend the auction or to finish it at any moment, for instance, after a particular bid.

- Auction.sol#L396

Recommendation

It is recommended to add an upper limit for the auction time and forbid decreasing the auction length.

Client's commentary

> MixBytes: auctions now can only be prolonged by new bids above determined `incrementalAmount`. Function `setPoolAuctionEnd` is removed.

L-12	The `amount` parameter is not validated in `increaseBid(address pool, uint256 amount)`
Severity	Low
Status	Fixed in 2d36634e

Description

The `increaseBid(address pool, uint256 amount)` function doesn't validate `amount`, zero amount can be passed to the function.

Recommendation

We recommend adding:

```
require(amount > 0, "");
```

Client's commentary

Fixed

2.5 Appendix

1 Monitoring Recommendation

The project contains smart contracts that require active monitoring. For these purposes, it is recommended to proceed with developing new monitoring events based on Forta (https://forta.org) with which you can track the following exemplary incidents:

- The pool has changed the state (from `Active` to `Warning`), especially check for wrong transitions (for instance, from `Closed` to `Default`)
- The pool has changed the price very much (the ratio of `cash` and `totalSupply`)
- Pool or Auction is closed not by the owner

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes