Smart Contract Security Audit Report

Clearpool USDX

1. Contents

2. General Information

This report contains information about the results of the security audit of the Clearpool (hereafter referred to as "Customer") USDX smart contracts, conducted by Decurity in the period from 04/29/2024 to 05/03/2024.

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/usdx. Initial review was done for the commit 452761ffacca8b949a9ecf284e3f7d81321708c8 and the re-testing was done for the commit 84667913e875346c5e30a8d5f3223aa979d48d72.

2.3. Threat Model

The assessment presumes the actions of an intruder who might have the capabilities of any role (an external user, token owner, token service owner, or a contract). The risks of centralization were not taken into account at the Customer's request.

The main possible threat actors are:

- Users,
- Protocol owner,
- Pool managers.

The table below contains sample attacks that malicious attackers might carry out.

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 a single high-risk issue, which has been fixed and re-tested in the course of the work.

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

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

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of May 3, 2023.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
A malicious manager can drain all rewards from factory (except FLARE)	TreasuryYield.sol	**High**	Fixed
Users cannot withdraw from pools with KYC	TreasuryYield.sol	**High**	Fixed
No storage gap for upgradeable contract	PoolFactoryInfo.sol, TreasuryYieldInfo.sol	**Medium**	Fixed
Incorrect reward amount calculations after reward rate update	PoolFactory.sol, TreasuryYield.sol	**Medium**	Fixed
Tokens with high decimals are not supported	utils.sol	**Low**	Acknowledged
Unused YEAR constant	Utils.sol, RewardAsset.sol	**Info**	Fixed
Wrong event data	PoolFactory.sol	**Info**	Fixed

Issue	Contract	Risk Level	Status
Invalid NatSpec	PoolFactory.sol	**Info**	Fixed
Redundant SafeCast	RewardAsset.sol, TreasuryYieldPool.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. A malicious manager can drain all rewards from factory (except FLARE)

Risk Level: High

Status: Fixed in commit 127b977c.

Contracts:

- TreasuryYield.sol

Location: Lines: 105. Function: `setRewardAssetInfo`.

Description:

A malicious pool manager has the ability to add an extra reward asset with an arbitrarily large reward rate through the `setRewardAssetInfo` function, even if this reward asset is intended for use only in a different pool.

Following this, he could deposit a minor sum into the pool and instantly withdraw the rewards. This effectively drains all rewards from other pools (except `FLARE` token), as the factory doesn't differentiate between the reward amounts across different pools.

The key problem is that a malicious manager can add a reward asset, meant for use in another pool, to their visible pool and drain all the rewards because of the large rate. The factory contract, which holds all the reward assets, does not check the allocation of reward tokens to specific pools.

All the rules above do not fix this issue.

Remediation:

Consider implementing one of the next approaches to reducing the risk:

- Implementing a synchronized accounting between the factory and the pools
- Introducing a delay for the `setRewardAssetInfo`
- Rebalancing of the rewards in the event of the rate change (the old rewards will be calculated based on the old rate, but the factory can still be drained)

- Limiting the `rate` value (this can make it too hard to completely drain the pool but still, a malicious pool can steal some of the rewards)

5.2. Users cannot withdraw from pools with KYC

Risk Level: High

Status: Fixed in commit 90c6c835.

Contracts:

- TreasuryYield

Location: Lines: 94. Function: `withdraw`.

Description:

To withdraw a deposit, a user needs to burn their shares in the `withdraw` function.

```
    function withdraw(
        uint256 _amount
    ) external nonReentrant nonZeroValue(_amount) {
        uint256 balance = balanceOf(msg.sender);
        if (_amount == type(uint256).max) {
            _amount = balance;
        }
        if (_amount > cash()) revert InsufficientLiquidity();

        _burn(msg.sender, _amount);
        asset.safeTransfer(msg.sender, _amount);
        emit Withdrawn(msg.sender, _amount);
    }
```

Due to the override of the `ERC-20` function `_update`, the burn address will be passed to the modifier `onlyEligible()`, which checks that the address is whitelisted when the pool has enabled KYC.

```
    function _update(
        address from,
        address to,
        uint256 amount
    ) internal override onlyEligible(to) {
        if (from != address(0)) _accrueReward(from);

        if (to != address(0)) _accrueReward(to);

        super._update(from, to, amount);
    }
```

```
    modifier onlyEligible(address _addr) {
        if (kycRequired && !poolFactory.whitelistedLenders(_addr))
            revert NotWhitelisted();
        _;
    }
```

However, the default zero address, which is used for burning tokens in ERC-20, cannot be whitelisted because of the nonZeroAddress check in the function setLenderStatus.

```
    function setLenderStatus(
        address _lender,
        bool status
    ) external onlyOwner nonZeroAddress(_lender) {
        if (whitelistedLenders[_lender] == status) revert NotSameValue();
        whitelistedLenders[_lender] = status;
        emit WhitelistStatusChanged(_lender, status);
    }
```

Therefore, users cannot withdraw their deposits due to the burn address not being whitelisted.

Remediation:

Consider adding additional check for zero address.

5.3. No storage gap for upgradeable contract

Risk Level: Medium

Status: Fixed in commit a01f36b8.

Contracts:

- PoolFactoryInfo,
- TreasuryYieldInfo

Description:

For upgradeable contracts, there must be storage gap to allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployment. Otherwise it may be very difficult to write new implementation code.

Without storage gap, the variable in child contract might be overwritten by the upgraded base contract if new variables are added to the base contract. This could have unintended and very serious

consequences to the child contracts, potentially causing loss of user fund or cause the contract to malfunction completely.

Remediation:

Consider adding appropriate storage gap at the end of upgradeable contracts such as the below.

```
uint256[50] private __gap;
```

5.4. Incorrect reward amount calculations after reward rate update

Risk Level: Medium

Status: Fixed in commit 5c385740.

Contracts:

- PoolFactory,
- TreasuryYield

Location: Function: `changeFlareRewardRate, setRewardAssetInfo`.

Description:

The reward rate, used in calculations of the amount of reward tokens for depositors, can be updated in the functions `changeFlareRewardRate`, `setRewardAssetInfo`. The number of reward tokens for each depositor is calculated only at the time of withdrawal. If a user does not withdraw rewards before an update to the rate, the tokens for the period prior to the update will also be calculated using the new rate. This could result in an inaccurate calculation of the reward.

Remediation:

Consider accumulating rewards for all lenders at the current rate before applying the new reward rate.

5.5. Tokens with high decimals are not supported

Risk Level: Low

Status: Acknowledged

Contracts:

- utils.sol

Location: Function: `toWei, fromWei`.

Description:

The functions `toWei` and `fromWei` contain a hardcoded value of `18` as the maximum for token decimals.

However, certain tokens possess more than 18 decimals (for example, `YAM-V2` has 24).

This could lead to unexpected reverts during the accrual and withdrawal of rewards due to underflow.

```
   function toWei(
        uint256 amount_,
        uint256 decimals
    ) internal pure returns (uint256) {
        return amount_ * 10 ** (18 - decimals);
    }

    function fromWei(
        uint256 amount_,
        uint256 decimals
    ) internal pure returns (uint256) {
        return amount_ / 10 ** (18 - decimals);
    }
}
```

Remediation:

Consider supporting tokens with high decimals.

5.6. Unused YEAR constant

Risk Level: Info

Status: Fixed in commit [a01f36b8](a01f36b8).

Description:

The constant `YEAR` is defined in contracts `UtilsGuard` and `RewardAsset` but never used.

Remediation:

Remove the constant.

5.7. Wrong event data

Risk Level: Info

Status: Fixed in commit dcb92990.

Contracts:

- PoolFactory.sol

Location: Lines: 160. Function: `createPool`.

Description:

The function `createPool` emits the event `PoolCreated` with incorrect data. Rather than emitting the manager's address, it emits `msg.sender`.

```
emit PoolCreated(
            _params.asset,
            address(treasuryYieldPool),
            msg.sender,
            _params.kycRequired
        );
```

Remediation:

Consider fixing the event.

5.8. Invalid NatSpec

Risk Level: Info

Status: Fixed in commit 5851a836.

Contracts:

- PoolFactory.sol

Location: Lines: 88.

Description:

NatSpec of struct `CreatePoolVars` contains param `symbol`, however, the struct does not contain it.

```
   /**
     * @param name the name of the pool token
     * @param symobl the symbol of the pool token // @audit No such param
```

```
     * @param asset the address of the asset used in the pool
     * @param manager the address of the manager
     * @param kycRequired the flag that shows if kyc is enabled or not for the
pool
     */
    struct CreatePoolVars {
        string name;
        address asset;
        address manager;
        bool kycRequired;
    }
```

Remediation:

Consider fixing NatSpec.

5.9. Redundant SafeCast

Risk Level: Info

Status: Fixed in commit 5c385740.

Contracts:

- RewardAsset.sol,
- TreasuryYieldPool.sol

Description:

The `TreasuryYield` and `RewardAsset` contracts import OpenZeppelin's `SafeCast` library. They use it for `uint256` and `int256` variable types:

```
using SafeCast for uint256;
using SafeCast for int256;
```

However, functions from this library are not used anywhere in the code.

Remediation:

Consider removing `SafeCast` library from contracts.

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.