CyberDEX - Public Sale

Audit Report

Version 1.1

Zigtur

June 29, 2024

CyberDEX - Public Sale - Audit Report

Zigtur

June 29, 2024

Prepared by: Zigtur

Table of Contents

Introduction

Disclaimer

A smart contract security review cannot guarantee the complete absence of vulnerabilities. This effort, bound by time, resources, and expertise, aims to identify as many security issues as possible. However, there is no assurance of 100% security post-review, nor is there a guarantee that the review will uncover all potential problems in the smart contracts. It is highly recommended to conduct subsequent security reviews, implement bug bounty programs, and perform on-chain monitoring.

About Zigtur

Zigtur is an independent blockchain security researcher dedicated to enhancing the security of the blockchain ecosystem. With a history of identifying numerous security vulnerabilities across various protocols in public audit contests and private audits, **Zigtur** strives to contribute to the safety and reliability of blockchain projects through meticulous security research and reviews. Explore previous work here or reach out on X @zigtur.

About CyberDEX

CyberDEX is a decentralised perpetual swaps trading platform functioning on Optimism. The exchange is powered by Synthetix which - by its unique model - allows traders to tap into huge liquidity pools and carry out trades with minimum slippage and market impact.

The project will soon launch a public sale which needs a security review.

Security Assessment Summary

Review commit hash - cd6027103ea0c9b5d2116906201fc73804f2ce99

Fixes review commit hash - 69ec796c2435059470d944a55ef19317f318125e

Deployment chains

- All EVM chains/rollups

Scope

The following smart contracts are in scope of the review:

- PublicSale.sol
- PublicSale.s.sol

Risk Classification

	Impact: High	**Impact:** Medium	**Impact:** Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Issues

HIGH-01 - USDT and USDC addresses are inverted in deployment script

Description

The `deploy` function creates two variables `USDT` and `USDC` and use them for deploying the `PublicSale` contract.

However, these two variables are incorrect. The `USDT` variable is initialized with USDC's mainnet address, and vice-versa.

Impact

User wanting to use USDT will use USDC instead, and vice-versa.

Code snippet

Scope:

- PublicSale.s.sol#L18-L19

`PublicSaleScript:deploy` function creates `USDT` and `USDC` variables:

```
1     function deploy() public returns(PublicSale sale){
2
3         address DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
4         address USDT = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48; //
              @POC: USDC mainnet address
5         address USDC = 0xdAC17F958D2ee523a2206206994597C13D831ec7; //
              @POC: USDT mainnet address
```

`USDT` variable is initialized with USDC mainnet address: 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48.

`USDC` variable is initialized with USDT mainnet address: 0xdAC17F958D2ee523a2206206994597C13D831ec7.

Recommendation

Switch `USDT` and `USDC` addresses to get the expected addresses.

A patch implementing this fix is available in Appendix.

Resolution

CyberDEX team: Fixed.

Zigtur: Fix reviewed and approved.

MEDIUM-01 - `transferTokensWithoutProof` and `claim` may not work with account abstraction wallets

Description

`transferTokensWithoutProof` and `claim` use `transfer` to send ETH value to the buyer (a.k.a claimer) address.

However, `transfer` only forwards 2300 gas. This may break integration with account abstraction wallets.

Impact

Account abstraction wallets may not be usable. Safe wallets are not impacted.

Code snippet

Scope:

- PublicSale.sol#L328
- PublicSale.sol#L397

Both `claim` and `transferTokensWithoutProof` use `transfer` to send ETH. The following snippet is taken from the `claim` function:

```
    function claim(
        address _claimer,
        uint120 _filledTokens,
        uint120 _unusedUsdc,
        uint120 _unusedUsdt,
        uint120 _unusedDai,
        uint120 _unusedEth,
        bytes32[] memory _proof
    ) external onlyInit onlyEnd {
        // ...

        if (_unusedEth > 0) payable(_claimer).transfer(_unusedEth);

        //...
    }
```

Recommended mitigation

Consider using `call` instead of `transfer`.

Note that using `call` instead of `transfer` will make these functions reentrant. However, `transferTokensWithoutProof` implements access control and the two functions make external calls in a safe state. This makes any reentrancy vector inexploitable for these two functions.

Resolution

CyberDEX team: Acknowledged. Safe wallets being compatible with the protocol is enough.

Zigtur: Acknowledged.

MEDIUM-02 - No sanity check for `priceFeed.latestRoundData` return values

Description

Currently, there are not validation of the return values from `latestRoundData` of Chainlink oracle.

```
    function _getLatestPrice() internal view returns (int256) {
        (, int256 _price,,,) = priceFeed.latestRoundData();
        return _price;
    }
```

Recommended mitigation

When fetching a price from a Chainlink oracle, consider the following checks:

- Consider basic sanity checks on the `answer` and the `updatedAt` values

```
require(answer > 0);
require(updatedAt != 0);
require(updatedAt <= block.timestamp);
```

- Consider a check for staleness with a reasonable `timeout`

```
require(block.timestamp - updatedAt < timeout);
```

The timeout can be configured through `setPriceFeed` by adding an argument

- Consider checking the oracle price range (`minAnswer` and `maxAnswer`)

A patch is available in Appendix.

Resolution

CyberDEX: Fixed. Sanity checks including a timeout check have been implemented.

Zigtur: Fix reviewed and approved.

LOW-01 - `createBuyOrder` makes the assumption that 1 stablecoin is 1 USD

Description

Scope:

- PublicSale.sol#L245-L252

`createBuyOrder` makes the assumption that every unit of stablecoin has a value of 1 USD.

This is not always the case. For example, this may be profitable to users which may use DAI instead of USDC when DAI is less expensive.

Recommendation

Fixing this issue would require retrieving the price for oracles such as Chainlink to determine the accurate amount of each stablecoin.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

LOW-02 - ERC20 `transfer` and `transferFrom` return values are not checked

Description

The current codebase uses `ERC20.transfer` and `ERC20.transferFrom`.

This can be problematic when interacting with tokens that are non ERC-20 compliant.

Recommendation

It is good practice to use the `SafeERC20` library from OpenZeppelin for these external interactions.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

LOW-03 - `createBuyOrderEth` makes the assumption that `_ethPrice` is 8 decimals

Description

The `createBuyOrderEth` retrieves ETH price from Chainlink to calculate the amount of tokens to order. These calculations require the ETH price to be based on 8 decimals.

However, the ETH price from Chainlink is never ensured to be based on 8 decimals.

Code snippet

Scope:

- PublicSale.sol#L273

The `createBuyOrderEth` determines the amount of tokens to order based on the ETH price:

```
1     function createBuyOrderEth() external payable onlyInit {
2         // ...
3
4         int256 _ethPrice = _getLatestPrice();
5         uint256 _tokens = (uint256(_ethPrice) * msg.value) / (price * 1
              e2);
6
7         // ...
8     }
```

Recommended mitigation

Add a check in `setPriceFeed` to ensure that the configured price oracle returns prices with 8 decimals.

A patch is available in Appendix.

Resolution

CyberDEX team: Acknowledged. The configured price feed is based on 8 decimals as expected.

Zigtur: Acknowledged.

INFO-01 - Anyone can claim on behalf of a claimer

Description

Scope:

- PublicSale.sol#L301

The `claim` function allows anyone to claim funds for a claimer.

The only required data is the Merkle Tree entry.

Recommendation

If this behavior is not expected, consider ensuring that `claimer == msg.sender`.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-02 - There can be only one MerkleTree entry per claimer

Description

Scope:

- PublicSale.sol#L312-L313

A claimer should not have multiple entries in the Merkle Tree. When claiming one entry, the receipt associated to this claimer is marked as `claimed = true`.

Any other claim for this claimer will be unusable as the call will revert with error `AlreadyClaimed()`.

Recommendation

Do not create multiple entries in the Merkle Tree for a single adress.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-03 - `supplyOrdered` is not capped

Description

The `supplyOrdered` value is increased when an order is created. The public sale may lead the `supplyOrdered` to be greater than the minted supply.

Recommendation

The `supplyOrdered` could be capped to the total supply of the token.

Note that this has no significant impact as the distribution is handled by the admin, and some tokens may be unfilled.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-04 - Non-trivial condition in `createBuyOrder` and `createBuyOrderEth`

Description

Scope:

- PublicSale.sol#L233
- PublicSale.sol#L265

`createBuyOrder` and `createBuyOrderEth` implement a check to ensure that the provided amount is not zero.

However, it checks `amount < 1` instead of checking `amount == 0`.

Recommendation

Consider using `== 0` instead of `< 1` for readability purposes.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-05 Duplicated code in `setClaimRoot`

Description

Scope:

- PublicSale.sol#L182-L185

`setClaimRoot` checks that the current timestamp is not before `start`.

However, this check is already executed through the `onlyEnd` modifier.

```
    function setClaimRoot(bytes32 _newRoot) public onlyOwner onlyEnd {
        if (block.timestamp < start) { // @POC: Useless check
            revert SaleNotStarted(block.timestamp, start);
        }
```

Recommendation

Consider removing the timestamp check from `setClaimRoot` as it is already done in `onlyEnd`.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-06 - Prefer Ownable2Step over Ownable

Description

Scope:

- PublicSale.sol#L61

The `PublicSale` contract inherits `Ownable` contract from OpenZeppelin.

Recommendation

Consider using `Ownable2Step` instead of `Ownable` to improve security and avoid human errors.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-07 - Incorrect comment for `_price` parameter in `initialise`

Description

Scope:

- PublicSale.sol#L163

A comment on the `initialise` function indicates the following about the `_price` parameter:

```
/// @param _price The `_usdc` payment value of each `_token`.
```

However, this value is not the `_usdc` payment value as other tokens can be used.

Recommendation

Consider fixing the comment. The following comment can be used:

```
/// @param _price The USD payment value of each `_token` with a 6
    decimals basis.
```

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-08 - Incorrect comment for `withdraw` and `withdrawEth`

Description

`withdraw` and `withdrawEth` comments indicate that they allow withdrawing "less than the balance".

```
/// @notice Withdraw any amount(less than the balance) of an ERC20
    token from this contract to a receiver
```

However, they allow withdrawing "less than or equal to the balance".

Recommendation

Consider fixing the comments. The following comment can be used:

```
/// @notice Withdraw any amount (less than or equal to the balance) of
    an ERC20 token from this contract to a receiver
```

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-09 - Inaccurate error in `transferTokens`

Description

Scope:

- PublicSale.sol#L357

When arrays length mismatch in `transferTokens`, the transaction reverts with a `NoAccess()` error.

Recommendation

Consider creating an `ArrayLengthMismatch()` error and reverting with it when arrays length mismatch.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

INFO-10 - Admin must be trusted

Description

The `admin` address has important privileges over the contract.

If malicious, this address can drain all funds from the contract.

Recommendation

None.

Resolution

CyberDEX team: Acknowledged.

Zigtur: Acknowledged.

Appendix

HIGH-01 - Patch

The following patch can be applied through `git apply`:

```
diff --git a/script/PublicSale.s.sol b/script/PublicSale.s.sol
index 3cb6eb8..3d970e2 100644
--- a/script/PublicSale.s.sol
+++ b/script/PublicSale.s.sol
@@ -15,8 +15,8 @@ contract PublicSaleScript is Script {
         function deploy() public returns(PublicSale sale){

                 address DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
-                address USDT = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;
-                address USDC = 0xdAC17F958D2ee523a2206206994597C13D831ec7;
+                address USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;
+                address USDT = 0xdAC17F958D2ee523a2206206994597C13D831ec7;
                 address TOKEN = 0xA0084063Ea01D5F09E56EF3fF6232A9e18B0BACD;
                 address PRICE_FEED = 0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419;
                 address ADMIN = 0xd3313105622B0aA4EDa599E62e0673d4D4DA4b16;
```

MEDIUM-02 - Patch

The following patch can be applied through `git apply`:

```
diff --git a/script/PublicSale.s.sol b/script/PublicSale.s.sol
index 3cb6eb8..c773486 100644
--- a/script/PublicSale.s.sol
+++ b/script/PublicSale.s.sol
@@ -19,6 +19,7 @@ contract PublicSaleScript is Script {
         address USDC = 0xdAC17F958D2ee523a2206206994597C13D831ec7;
         address TOKEN = 0xA0084063Ea01D5F09E56EF3fF6232A9e18B0BACD;
         address PRICE_FEED = 0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419
             ;
+        uint256 PRICE_TIMEOUT = 2 hours; // ETH/USD price feed has a 1
    hour heartbeat
         address ADMIN = 0xd3313105622B0aA4EDa599E62e0673d4D4DA4b16;
         address TREASURY = 0xd3313105622B0aA4EDa599E62e0673d4D4DA4b16;
          sale = new PublicSale(
@@ -27,6 +28,7 @@ contract PublicSaleScript is Script {
             USDC,
             TOKEN,
             PRICE_FEED,
+            PRICE_TIMEOUT,
             ADMIN,
             TREASURY
         );
diff --git a/src/PublicSale.sol b/src/PublicSale.sol
index a7f27bc..e8c18a6 100644
--- a/src/PublicSale.sol
+++ b/src/PublicSale.sol
@@ -24,6 +24,8 @@ error ZeroAddress();
 error ZeroAmount();
 error MoreThanBalance();
 error NoAccess();
+error IncorrectOracleAnswer();
+error StaleEthPrice();

 /*//////////////////////////////////////////////////////////////
                            INTERFACES
@@ -83,6 +85,8 @@ contract PublicSale is IPublicSale, Ownable {
     // Chainlink Aggregator interface
     IAggregatorV3 public priceFeed;

+    uint256 public priceTimeout;
+
     /// When the sale begins.
     uint40 public start;
     /// How long the sale goes for.
@@ -142,6 +146,7 @@ contract PublicSale is IPublicSale, Ownable {
         address _usdc,
         address _token,
```

```
46            address _priceFeed,
47  +         uint256 _priceTimeout,
48            address _admin,
49            address _treasury
50        ) {
51  @@ -150,6 +155,7 @@ contract PublicSale is IPublicSale, Ownable {
52            usdc = _usdc;
53            token = IERC20(_token);
54            priceFeed = IAggregatorV3(_priceFeed);
55  +         priceTimeout = _priceTimeout;
56            admin = _admin;
57            treasury = _treasury;
58
59  @@ -189,10 +195,11 @@ contract PublicSale is IPublicSale, Ownable {
60
61        /// @notice allows the owner to set the priceFeed contract's
              address
62        /// @param _priceFeed address of the new priceFeed contract
63  -     function setPriceFeed(address _priceFeed) external {
64  +     function setPriceFeed(address _priceFeed, uint256 _priceTimeout)
      external {
65            if (msg.sender != admin) revert NoAccess();
66            if (_priceFeed == address(0)) revert ZeroAddress();
67            priceFeed = IAggregatorV3(_priceFeed);
68  +         priceTimeout = _priceTimeout;
69            emit PriceFeedUpdate(_priceFeed);
70        }
71
72  @@ -433,7 +440,11 @@ contract PublicSale is IPublicSale, Ownable {
73
74        /// @notice get the latest price for eth from Chainlink's
              Aggregator PriceFeed
75        function _getLatestPrice() internal view returns (int256) {
76  -         (, int256 _price,,,) = priceFeed.latestRoundData();
77  +         (, int256 _price,, uint256 updatedAt,) = priceFeed.
      latestRoundData();
78  +
79  +         if (_price <= 0) revert IncorrectOracleAnswer();
80  +         if (block.timestamp - updatedAt > priceTimeout) revert
      StaleEthPrice();
81  +
82            return _price;
83        }
```

LOW-03 - Patch

The following patch can be applied through `git apply`. It adds a check to ensure that the price feed returns 8 decimals prices.

```
diff --git a/src/PublicSale.sol b/src/PublicSale.sol
index a7f27bc..fd50680 100644
--- a/src/PublicSale.sol
+++ b/src/PublicSale.sol
@@ -193,6 +193,7 @@ contract PublicSale is IPublicSale, Ownable {
          if (msg.sender != admin) revert NoAccess();
          if (_priceFeed == address(0)) revert ZeroAddress();
          priceFeed = IAggregatorV3(_priceFeed);
+         require(priceFeed.decimals() == 8);
          emit PriceFeedUpdate(_priceFeed);
      }
```