Coinsult

Advanced Manual Smart Contract Audit

Project: Cybermine Finance

Website: https://www.cybermine.finance/

Low-Risk	Medium-Risk	High-Risk
7 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

NA

Disclaimer

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Tokenomics

NA

Source Code

Coinsult was comissioned by Cybermine Finance to perform an audit based on the following smart contract:

https://github.com/CyberminFinance/CME/blob/main/ProsperityCoin%20new.sol

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

7 low-risk code
issues found

Medium-Risk

0 medium-risk code
issues found

High-Risk

0 high-risk code
issues found

The detailed report continues on the next page...

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Unused commented code

```
function calcPledgeReward(uint256 amount, uint startTime) public view returns(uint256) {
     // amount *  (block.timestamp - start)/5 * (5024835580/10512000000);
     //
     return amount *  (block.timestamp - startTime) * 5024835580/52560000000;
}
```

Recommendation

Remove commented code for readability.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Contract contains Reentrancy vulnerabilities

Additional information: This combination increases risk of malicious intent. While it may be justified by some complex mechanics (e.g. rebase, reflections, buyback).

More information: Slither

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal virtual {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(_balances[from] &gt;= amount, "ERC20: transfer amount exceeds balance");


    if(_enableWhiteList){
        //
        if(from == _pool){
            // 买
            require(_whiteList[to].updateTime &gt; 0 &amp;&amp; _whiteList[to].isWhite, "CME: to add
        }else if(to == _pool){
            //
            require(_whiteList[from].updateTime &gt; 0 &amp;&amp; _whiteList[from].isWhite, "CME: fr
        }
    }

     beforeTokenTransfer(from, to, amount);
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if mgs.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function calcPledgeReward(uint256 amount, uint startTime) public view returns(uint256) {
     // amount *  (block.timestamp - start)/5 * (5024835580/10512000000);
     //
     return amount *  (block.timestamp - startTime) * 5024835580/52560000000;
}
```

Recommendation

Do not use `block.timestamp`, now or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
      reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Too many digits

Literals with many digits are difficult to read and review.

```
_mint(address(this), 1000000000000000000000000000);
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```
contract MyContract{
    uint 1_ether = 1000000000000000000;
}
```

While `1_ether` looks like `1 ether`, it is `10 ether`. As a result, it's likely to be used incorrectly.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

No zero address validation for some functions

Detect missing zero address validation.

```
function updateNationalTreasuryAddress(address nationalTreasuryAddress) public onlyOwner {
    _nationalTreasuryAddress = nationalTreasuryAddress;
}
```

Recommendation

Check that the new address is not zero.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

Bob calls `updateOwner` without specifying the `newOwner`, soBob loses ownership of the contract.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function updateEnableWhiteList(bool enable) public onlyOwner {
    _enableWhiteList = enable;
}

function setWhileAddress(address whileAddress, bool isWhite) public onlyOwner {
    _whiteList[whileAddress].updateTime = block.timestamp;
    _whiteList[whileAddress].isWhite = isWhite;
}

function deleteWhileAddress(address whileAddress) public onlyOwner {
    require(_whiteList[whileAddress].updateTime != 0, "CME: input address is not in while list");
    delete _whiteList[whileAddress];
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

`updateOwner()` has no event, so it is difficult to track off-chain changes in the buy price.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Boolean equality

Detects the comparison to boolean constants.

```
function setPool(address pool) public onlyOwner {
    require(pool != address(0) &amp;amp;&amp;amp; isContract(pool) == true, "CME: invalid pool address");
    _pool = pool;
    asyncPair();
}
```

Recommendation

Remove the equality to the boolean constant.

Exploit scenario

```
contract A {
    function f(bool x) public {
        // ...
        if (x == true) { // bad!
           // ...
        }
        // ...
    }
}
```

Boolean constants can be used directly and do not need to be compare to `true` or `false`.

Owner privileges

- 🟢 Owner cannot set fees higher than 25%
- 🟢 Owner cannot pause trading
- 🟢 Owner cannot change max transaction amount
- ⚠️ Owner can set a new pool
- ⚠️ Owner can change the EXO and EVO address
- ⚠️ Owner can enable whitelist mode, only whitelisted wallets can transfer

Extra notes by the team

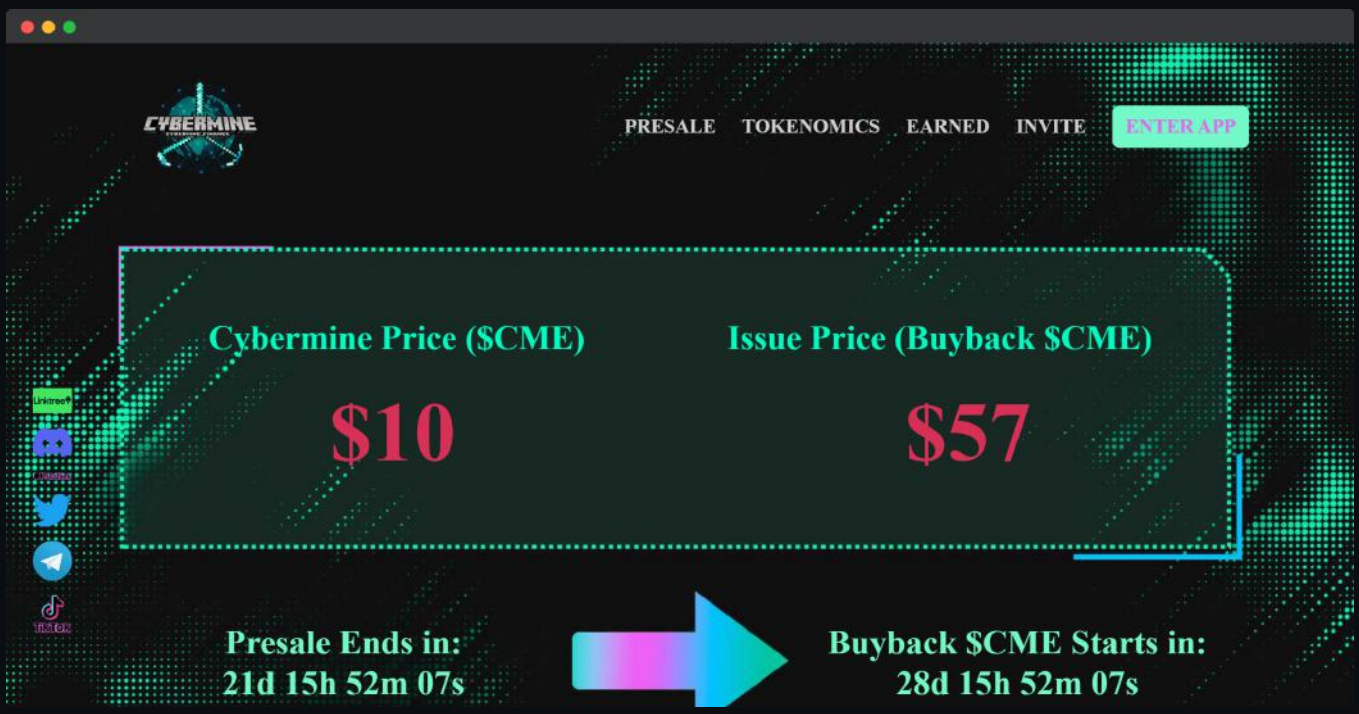
People can swap EXO to CME, whereafter their EXO gets burned.

Contract Snapshot

```
contract ProsperityCoin is  Context, IERC20, IERC20Metadata, Ownable {
struct UserInfo{
    uint updateTime;
    bool isWhite;
}
struct PledgeInfo{
    uint updateTime;
    uint256 amount;
}
mapping(address =&gt; UserInfo) public _whiteList;
mapping(address =&gt; PledgeInfo) public _pledges;

mapping(address =&gt; uint256) private _balances;
address public _pool;
uint8 public _decimals = 18;

mapping(address =&gt; mapping(address =&gt; uint256)) private _allowances;

uint256 private _totalSupply;

string private _name;
string private _symbol;

IERC20 public _exoContract;
IERC20 public _evoContract;
IPancakePair public _poolContract;
address public _poolAddress;
address public _nationalTreasuryAddress;
bool public _enableWhiteList;

// FLDI
uint256 public _FLDI;
// EXGE
uint256 public _EXGE;
// M
uint256 public _M;
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



- Mobile Friendly
- Does not contain jQuery errors
- SSL Secured
- No major spelling errors

Project Overview

Not KYC verified by Coinsult

Cybermine Finance

Audited by Coinsult.net

Date: 16 August 2022

✔ Advanced Manual Smart Contract Audit