Coinsult

Advanced Manual Smart Contract Audit

Project: F8 Dao

Website: https://f8dao.io

Low-Risk	Medium-Risk	High-Risk
6 low-risk code issues found	3 medium-risk code issues found	0 high-risk code issues found

Contract Address

0x54fdd659ef83708c6dfd460fb309113db688a9df

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

-

Source Code

Coinsult was comissioned by F8 Dao to perform an audit based on the following smart contract:

https://bscscan.com/address/0x54fdd659ef83708c6dfd460fb309113db688a9df#code

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

6 low-risk code
issues found

Medium-Risk

3 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.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
if(player.lastWithdrawTime.add(WITHDRAW_INTERVAL) &gt;= block.timestamp) {
    refReward = 0;
}
```

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.

```
if(totalInvestValue &gt;= 10000 &amp;&amp; player.refsCount &gt;= 13 &amp;&amp; _getTeamValidMemberC
    level = 3;
} else if (totalInvestValue &gt;= 2000 &amp;&amp; player.refsCount &gt;= 8 &amp;&amp; _getTeamValidM
    level = 2;
} else if (totalInvestValue &gt;= 1000 &amp;&amp; player.refsCount &gt;= 5 &amp;&amp; _getTeamValidM
    level = 1;
}
```

Recommendation

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

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function activation(address _ref) payable external {

    require(players[_ref].active, "Warning: ref must be activated.");
    require(block.timestamp &gt;= START_TIME, "Warning: Activity not started.");
    require(_ref != _msgSender(), "Warning: Referal can't refer to itself.");
    require(!players[_msgSender()].active, "Warning: You have been activated.");

    Player storage player = players[_msgSender()];

    if (!player.active) {
        player.active = true;
        playersCount += 1;

        player.referrer = payable(_ref);
        players[_ref].refsCount += 1;
        players[_ref].refsList.push(_msgSender());
    }

    activationList[_msgSender()] =  ref;
```

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.

Conformance to Solidity naming conventions

Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
contract Cards {

    uint256[4] internal CARD = [0, 1, 2, 3];
    uint256[4] internal CARD_VALUES = [100, 500, 2000, 5000];
    uint256[4] internal CARD_PROFITS = [10, 12, 15, 18];

    uint256[5] internal CARD_ALL_REWARDS_PERCENTS = [20, 15, 10, 8, 6];
    uint256[5][1] internal CARD_REWARDS_PERCENTS;

    constructor() {
        CARD_REWARDS_PERCENTS[0] = CARD_ALL_REWARDS_PERCENTS;
    }

}
```

Recommendation

Follow the Solidity naming convention.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (`ERC20`).
- Allow _ at the beginning of the `mixed_case` match for private variables and unused parameters.

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

Redundant Statements

Detect the usage of redundant statements that have no effect.

```
struct Player{
    bool active;
    address payable referrer;
    Holding[] holdings;
    uint256 refsCount;
    address[] refsList;
    uint256 referralReward;
    uint256[4] accumulatives;
    uint256 lastWithdrawTime;
    uint256 teamCount;
    address[] teamList;
    uint256 playerTotalValue;
    uint256 playerWithdrawAmount;
    uint256 teamPerformance;
    uint256 teamProfit;
    uint256 lastSellTime;
    uint256 maxDailyWithdrawnRewardAmount;
    uint256 dailyWithdrawnRewardAmount;
```

Recommendation

Remove redundant statements if they congest code but offer no value.

Exploit scenario

```
contract RedundantStatementsContract {

    constructor() public {
        uint; // Elementary Type Name
        bool; // Elementary Type Name
        RedundantStatementsContract; // Identifier
    }

    function test() public returns (uint) {
        uint; // Elementary Type Name
        assert; // Identifier
        test; // Identifier
        return 777;
    }
}
```

Each commented line references types/identifiers, but performs no action with them, so no code will be generated for such statements and they can be removed.

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

Costly operations inside a loop

Costly operations inside a loop might waste gas, so optimizations are justified.

```
function buyCard(uint256 _card) payable external nonReentrant {

    require(players[_msgSender()].active, "Warning: You have not activated.");
    require(_card &lt; CARD_TYPES_COUNT, &quot;Warning: Invalid card number.&quot;);

    Player storage player = players[_msgSender()];

    uint256 price = getCurrentPirce(CARD_VALUES[_card]);

    holdingsCounter += 1;

    player.holdings.push(
        Holding({
            id: holdingsCounter,
            cardType: _card,
            cardValue: CARD_VALUES[_card],
            cardProfit: CARD_PROFITS[_card],
            buyTime: block.timestamp,
```

Recommendation

Use a local variable to hold the loop computation result.

Exploit scenario

```
contract CostlyOperationsInLoop{

    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }

    function good() external{
      uint local_variable = state_variable;
      for (uint i=0; i < loop_count; i++){
        local_variable++;
      }
      state_variable = local_variable;
    }
}
```

Incrementing `state_variable` in a loop incurs a lot of gas because of expensive `SSTORE`s, which might lead to an `out-of-gas`.

🟡 **Medium-Risk:** Should be fixed, could bring problems.

Requirements about TRX, on the BSC + Spelling error

```
function sellCard(uint256 holdingIndex) payable external {

    require(players[_msgSender()].active, "Warning: You have not activated.");
    Player storage player = players[_msgSender()];

    require(player.lastSellTime.add(SELL_INTERVAL) &lt; block.timestamp, &quot;Warning: Sell time in
    require(holdingIndex = holding.buyPrice, "Warning: TRX not enought.");

    if (holdingIndex &lt; (player.holdings.length - 1)) {
        player.holdings[holdingIndex] = player.holdings[(player.holdings.length - 1)];
```

Recommendation

Revise the contract to avoid blind copy issues from another chain. Also remove spelling errors like ‘enought’.

🟡 **Medium-Risk:** Should be fixed, could bring problems.

Spelling error

```
function serProfitCap(uint256 _amount) external onlyOperator {
    PROFIT_CAP = _amount;
}
```

Recommendation

serProfitCap -> setProfitCap

🟡 **Medium-Risk:** Should be fixed, could bring problems.

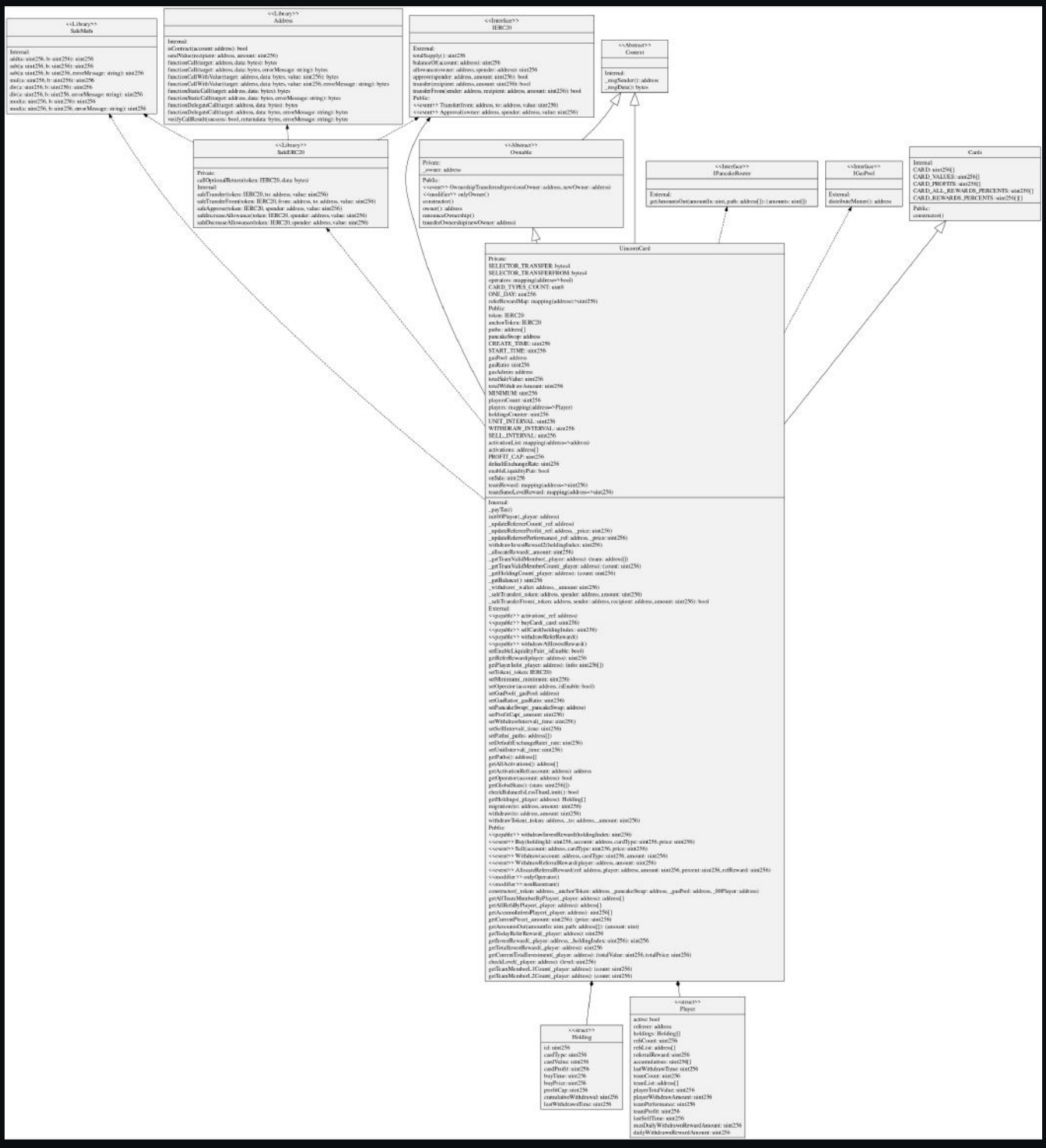
Potential Spelling error

```
contract UincornCard is Context, Ownable, Cards {

using SafeMath for uint256;
using SafeERC20 for IERC20;

struct Holding {
    uint256 id;
    uint256 cardType;
    uint256 cardValue;
    uint256 cardProfit;
```

Recommendation

UincornCard or UnicornCard?

Owner privileges



Extra notes by the team

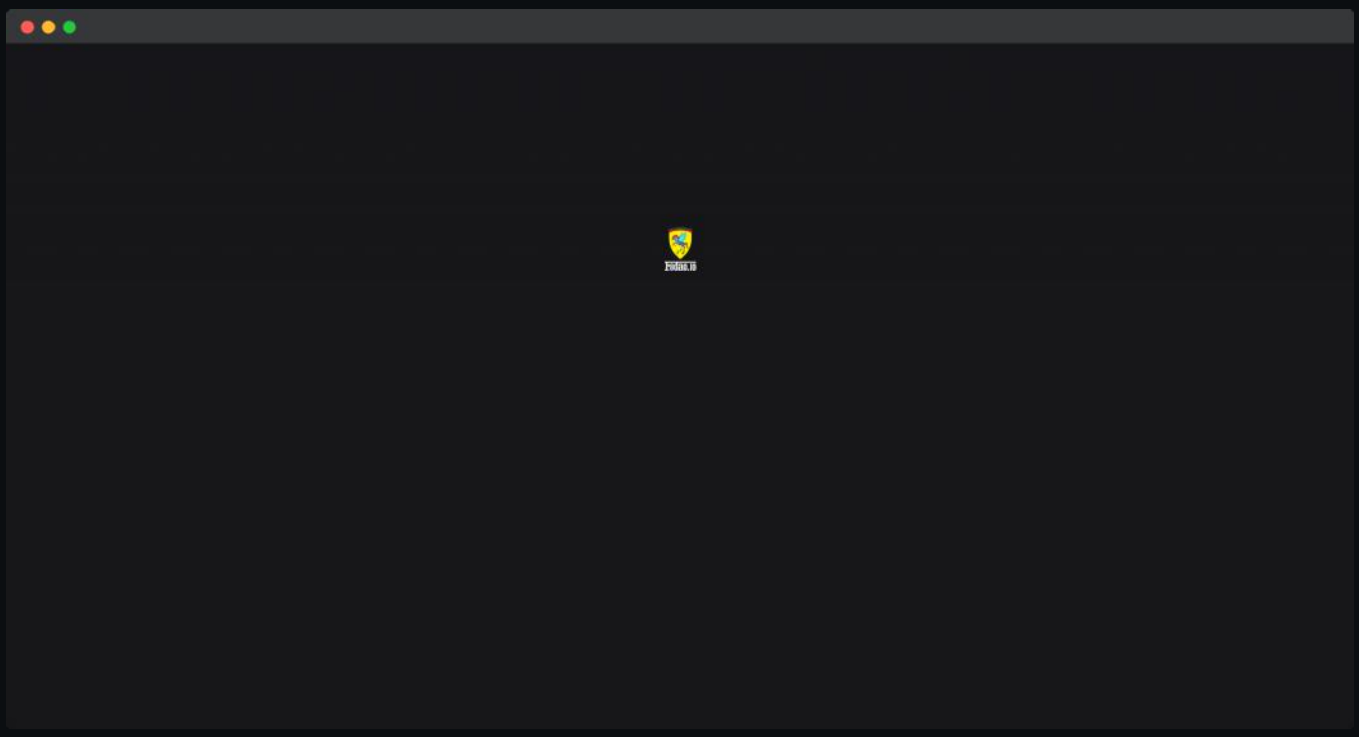
No notes

Contract Snapshot

```
contract UincornCard is Context, Ownable, Cards {

using SafeMath for uint256;
using SafeERC20 for IERC20;

struct Holding {
    uint256 id;
    uint256 cardType;
    uint256 cardValue;
    uint256 cardProfit;
    uint256 buyTime;
    uint256 buyPrice;
    uint256 profitCap;
    uint256 cumulativeWithdrawal;
    uint256 lastWithdrawnTime;
}
```

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