Coinsult

Advanced Manual Smart Contract Audit

Project: Metaverse MSN

Website: -No website-

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

Contract Address

0xaf51951df5782fa6eb529c173b10a973e4871f92

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

Rank	Address	Quantity (Token)	Percentage
1	0x4863a1c6500d297090d9dc351f37a4918d768fe7	1,054.920451257390099246	21.0984%
2	0x0de661868475d6f6da6861f1fa72dfd0b5eba33c	300	6.0000%
3	0xe07bb15eba6462e589f3bc3d1b3f6c25fca9c36c	200	4.0000%
4	0x5780f00117272d30cb8a13ae7b81c05975839fe0	186.728644013897455895	3.7346%
5	0xb68cfa1a68ead8cbe9300f2fc0859568d5287cf2	86.70557907935153395	1.7341%

Source Code

Coinsult was comissioned by Metaverse MSN to perform an audit based on the following smart contract:

https://bscscan.com/address/0xaf51951df5782fa6eb529c173b10a973e4871f92#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

6 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.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function Launch(uint256 _time) public onlyOwner {
    require(!isLaunch);
    isLaunch = true;
    if(_time == 0){
        startTime = block.timestamp;
    }else{
        startTime = _time;
    }
}
```

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.

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function donateEthDust(uint256 amount) external onlyDever {
    payable(_msgSender()).transfer(amount);
}
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Exploit scenario

```
contract ArbitrarySend{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls `setDestination` and `withdraw`. As a result he withdraws the contract's balance.

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

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function withdraw() public {
    uint256 fistBalance = fist.balanceOf(address(this));
    if (fistBalance &gt; 0) {
        fist.transfer(address(token), fistBalance);
    }
    uint256 tokenBalance = token.balanceOf(address(this));
    if (tokenBalance &gt; 0) {
        token.transfer(address(token), tokenBalance);
    }
}
```

Recommendation

Use `SafeERC20`, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in `MyBank`, `deposit` will not revert if the transfer fails, and an attacker can call `deposit` for free..

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

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
function _takeInviterFee(
    address sender,
    address recipient,
    uint256 tAmount
) private {
    address cur;
    address reciver;
    if (ammPairs[sender]) {
        cur = recipient;
    } else {
        cur = sender;
    }
    uint256 rate;
    uint256 rAmount = tAmount.div(1000).mul(10);
```

Recommendation

Consider ordering multiplication before division.

Exploit scenario

```
contract A {
    function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5; n = 10, interest = 2`, coins will be zero. If `(oldSupply * interest / n)` was used, `coins` would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

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

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setlpDivThres(uint256 _thres) public onlyOwner {
    lpTokenDivThres = _thres;
}
```

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.

Redundant Statements

Detect the usage of redundant statements that have no effect.

```
function _msgData() internal view virtual returns (bytes memory) {
    this;
    // silence state mutability warning without generating bytecode - see https://github.com/ethereu
    return msg.data;
}
```

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.

Owner privileges

- 🟢 Owner cannot set fees higher than 25%
- 🟢 Owner cannot pause trading
- 🟢 Owner cannot change max transaction amount
- 🟡 Owner can exclude from fees
- 🔴 Owner can blacklist addresses
- ⚠️ Owner can set minimum wallet value for inviter fee
- ⚠️ Owner can reset list of inviter addresses

Extra notes by the team

No notes

Contract Snapshot

```
contract MSN is ERC20 {
using SafeMath for uint256;

IUniswapV2Router02 public uniswapV2Router;
IUniswapV2Pair public  uniswapV2Pair;
address _tokenOwner;
address private _destroyAddress = address(0x000000000000000000000000000000000000dEaD);
address private fundAddress = address(0x05532F432956fe9ea53cD65B14194717003c2079);
address private ecoAddress = address(0xda0cbCC99628121c0Df3294f75b3B25642228003);
mapping(address =&gt; bool) private _isExcludedFromFees;
mapping(address =&gt; bool) private _isDelivers;
mapping(address =&gt; bool) public _isBot;
mapping(address =&gt; address) public inviter;
mapping(address =&gt; bool) public inviterBlack;
mapping(address =&gt; uint256) public inviterLockTime;
uint256 public inviterRequireLockTime;
uint256 public thresRefAmount;
bool public isLaunch = false;
uint256 public startTime;
bool public swapAndLiquifyEnabled = true;
bool private swapping = false;
```

Project Overview

Not KYC verified by Coinsult

Metaverse MSN

Audited by Coinsult.net

Date: 26 July 2022

Advanced Manual Smart Contract Audit