Coinsult

Advanced Manual Smart Contract Audit

Project: MIM Machine (Staking)
Website: https://mim-machine.com/

🟢 **Low-Risk**	🟡 **Medium-Risk**	🔴 **High-Risk**
8 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

Not deployed yet

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 MIM Machine (Staking) to perform an audit based on the following smart contract:

–

Contract not yet deployed

Manual Code Review

In this audit report we will highlight all these issues:

🟢 **Low-Risk**

8 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.

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 claimToken_M() public {
    User storage user = users[msg.sender];

    updateStakeBUSD_IP(msg.sender);
    uint256 tokenAmount = user.sM.unClaimedTokens;
    user.sM.unClaimedTokens = 0;

    _mint(msg.sender, tokenAmount);
    emit TokenOperation(msg.sender, "CLAIM", tokenAmount, 0);
}

function claimToken_T() public {
    User storage user = users[msg.sender];

    updateStakeToken_IP(msg.sender);
    uint256 tokenAmount = user.sT.unClaimedTokens;
    user.sT.unClaimedTokens = 0;

    _mint(msg.sender, tokenAmount);
    emit TokenOperation(msg.sender, "CLAIM", tokenAmount, 0);
}
```

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 getContractLaunchTime() public view returns (uint256) {
    return minZero(startTime, block.timestamp);
}
```

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.

```
function SET_SELL_LIMIT(uint256 value) external {
    require(msg.sender == ADMIN, "Admin use only");
    require(value &gt;= 40000);
    SELL_LIMIT = value * 1 ether;
}
```

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.

Functions that send Ether to arbitrary destinations

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

```
function stakeBUSD(uint256 _amount) public payable {
    require(block.timestamp &gt; startTime);
    require(_amount &gt;= MIN_INVEST_AMOUNT); // added min invest amount
    token.transferFrom(msg.sender, address(this), _amount); // added

    uint256 fee = _amount.mul(FEE).div(PERCENT_DIVIDER); // calculate fees on _amount and not msg.va

    token.transfer(DEV_POOL, fee);

    User storage user = users[msg.sender];

    if (user.sM.totalStaked == 0) {
        user.sM.checkpoint = maxVal(now, startTime);
        totalUsers++;
    } else {
        updateStakeBUSD_IP(msg.sender);
    }

    user.sM.lastStakeTime = now;
    user.sM.totalStaked = user.sM.totalStaked.add(_amount);
    totalBUSDStaked = totalBUSDStaked.add(_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 stakeToken(uint256 tokenAmount) public {
    User storage user = users[msg.sender];
    require(now &gt;= startTime, "Stake not available yet");
    require(
        tokenAmount &lt;= balanceOf(msg.sender),
        &quot;Insufficient Token Balance&quot;
    );

    if (user.sT.totalStaked == 0) {
        user.sT.checkpoint = now;
    } else {
        updateStakeToken_IP(msg.sender);
    }

    _transfer(msg.sender, address(this), tokenAmount);
    user.sT.lastStakeTime = now;
    user.sT.totalStaked = user.sT.totalStaked.add(tokenAmount);
    totalTokenStaked = totalTokenStaked.add(tokenAmount);
}
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function SET_MIN_INVEST_AMOUNT(uint256 value) external {
    require(msg.sender == ADMIN, "Admin use only");
    require(value &gt;= 5);
    MIN_INVEST_AMOUNT = value * 1 ether;
}

function SET_SELL_LIMIT(uint256 value) external {
    require(msg.sender == ADMIN, "Admin use only");
    require(value &gt;= 40000);
    SELL_LIMIT = value * 1 ether;
}
```

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.

```
function SET_MIN_INVEST_AMOUNT(uint256 value) external {
    require(msg.sender == ADMIN, "Admin use only");
    require(value &gt;= 5);
    MIN_INVEST_AMOUNT = value * 1 ether;
}

function SET_SELL_LIMIT(uint256 value) external {
    require(msg.sender == ADMIN, "Admin use only");
    require(value &gt;= 40000);
    SELL_LIMIT = value * 1 ether;
}
```

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.

```
function getContractBUSDBalance() public view returns (uint256) {
    // return address(this).balance;
    return token.balanceOf(address(this));
}
```

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 can change max transaction amount

⚠️ Owner can set minimum investment amount

⚠️ Owner can set sell limit
– Note from the MIM machine team: The sell limit will always be at least 40,000 ether

Extra notes by the team

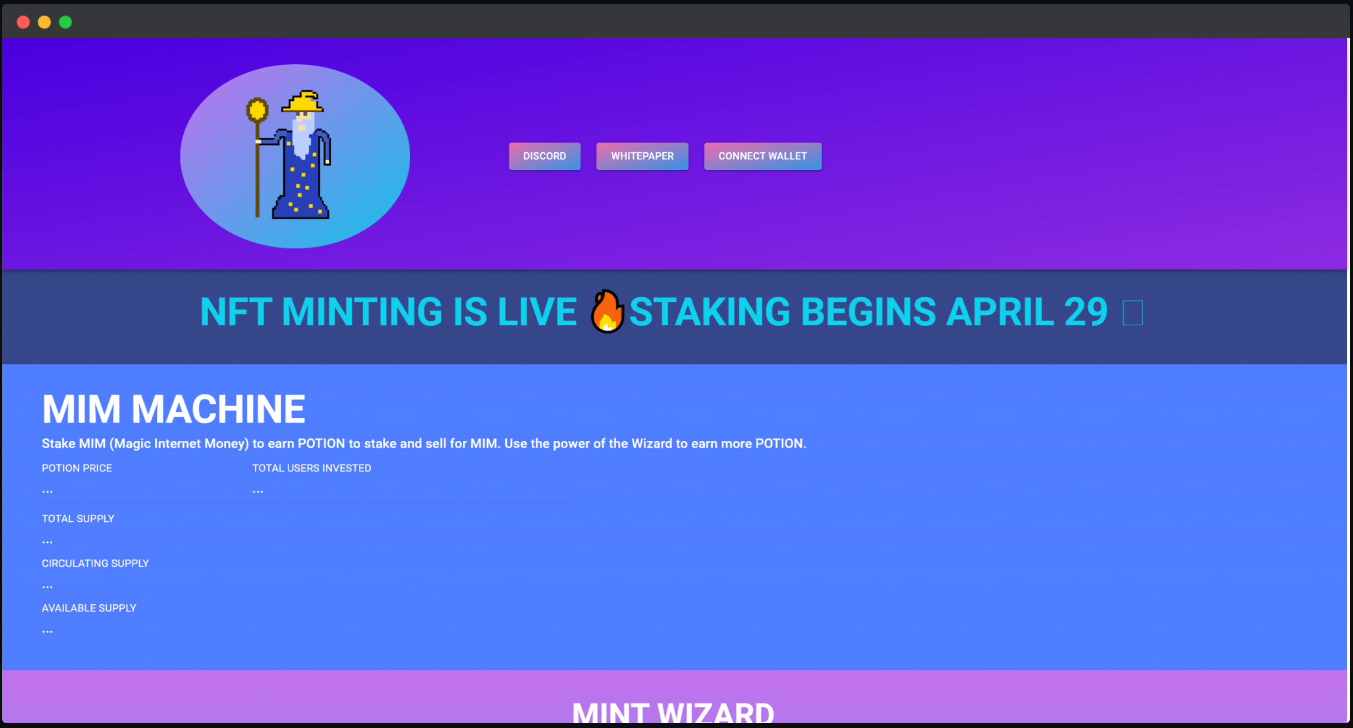
No notes

Contract Snapshot

```
contract MIMMachine is Token {
uint256 private startTime = now - 1 days;

address payable private ADMIN;
address payable private DEV_POOL;

uint256 public totalUsers;
uint256 public totalBUSDStaked;
uint256 public totalTokenStaked;

uint256 private constant FEE = 100; // 10% fee
uint256 private constant MANUAL_AIRDROP = 50000 ether; // marketing + giveaways

uint256 private constant PERCENT_DIVIDER = 1000;
uint256 private constant PRICE_DIVIDER = 1 ether;
uint256 private constant TIME_STEP = 1 days;
uint256 private constant TIME_TO_UNSTAKE = 7 days;


// Configurables
uint256 public MIN_INVEST_AMOUNT = 5 ether;
uint256 public SELL_LIMIT = 50000 ether;
uint256 public BUSD_DAILYPROFIT = 20; // 2%
uint256 public TOKEN_DAILYPROFIT = 40; // 4%

mapping(address =&gt; User) private users;
mapping(uint256 =&gt; uint256) private sold;

struct Stake {
    uint256 checkpoint;
    uint256 totalStaked;
    uint256 lastStakeTime;
    uint256 unClaimedTokens;
}

struct User {
    Stake sM; // staked BUSD
```

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.



NFT MINTING IS LIVE 🔥STAKING BEGINS APRIL 29

MIM MACHINE

Stake MIM (Magic Internet Money) to earn POTION to stake and sell for MIM. Use the power of the Wizard to earn more POTION.

POTION PRICE ...

TOTAL USERS INVESTED ...

TOTAL SUPPLY ...

CIRCULATING SUPPLY ...

AVAILABLE SUPPLY ...

MINT WIZARD

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

Note:

Pretty basic website, could use some more development.

Project Overview

● Not KYC verified by Coinsult