Coinsult

Request your audit at coinsult.net

Advanced Manual
Smart Contract Audit

September 5, 2022

Audit requested by

Meta Games Coin

0x3ADE350e05F631f946B6d2B1a2deAAF95DCB243a

Table of Contents

Audit Summary

Audit Scope

Project Name	Meta Games Coin
Website	https://metagamescoin.io/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x3ADE350e05F631f946B6d2B1a2deAAF95DCB243a
Audit Method	Static Analysis, Manual Review
Date of Audit	5 September 2022

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x3b49349e1b29c2b323891b21667c0f7708c326b6	979,011,990,943,059.484090605590206826	97.9012%
2	Null Address: 0x000...dEaD	18,009,000,005,226.709566366961386326	1.8009%
3	0xdb3d516b5dd0864b12b79377be5329f0c452ed27	1,000,000,000,000.017500000029845086	0.1000%
4	metagamescoin: Deployer	988,999,181,977.523657813940496937	0.0989%
5	0xf7fb34b0527abc9f08aa355f57b9ff9d5bd30d88	988,998,848,622.674972354752009177	0.0989%
6	0x4fe371df8b82518820eda1ed0c42bde133122fa4	999,000,020.067192212116422485	0.0001%
7	PancakeSwap V2: MGC 76	2,345,462.903648270128977213	0.0000%
8	0x046474cd376766e402a639d62db9d31b219862d0	1,992,796.001059153429408697	0.0000%
9	0x01096491328a9d10f6913be5332fd57392c6b9bf	1,591,531.423845202091197791	0.0000%
10	0xd8eea6e9850ee9a352096af07aa88526513f832e	1,000,000.000000004000000006	0.0000%

Source Code

Coinsult was comissioned by Meta Games Coin to perform an audit based on the following code:

https://bscscan.com/address/0x3ADE350e05F631f946B6d2B1a2deAAF95DCB243a#code

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

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.

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
🔵 Informational	0	0	0	0
🟢 Low-Risk	4	4	0	0
🟡 Medium-Risk	2	0	2	0
🔴 High-Risk	0	0	0	0

Privilege Overview

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	🟢 Owner cannot mint new tokens
Owner can blacklist?	🟢 Owner cannot blacklist addresses
Owner can set fees > 25%?	🔴 Owner can set the sell fee to 25% or higher
Owner can exclude from fees?	🔵 Owner can exclude from fees
Owner can pause trading?	🟢 Owner cannot pause the contract
Owner can set Max TX amount?	🔴 Owner can set max transaction amount

More owner priviliges are listed later in the report.

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

Unchecked transfer

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

```
function recoverBEP20(address tokenAddress, uint256 tokenAmount) public onlyOwner {
    // do not allow recovering self token
    require(tokenAddress != address(this), "Self withdraw");
    IERC20(tokenAddress).transfer(owner(), 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.

Divide before multiply

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

```
if(_burnFee != 0){
    spentAmount  = contractTokenBalance.div(totFee).mul(_burnFee);
    _tokenTransferNoFee(address(this), dead, spentAmount);
    totSpentAmount = spentAmount;
}

if(_walletFee != 0){
    spentAmount = contractTokenBalance.div(totFee).mul(_walletFee);
    _tokenTransferNoFee(address(this), feeWallet, spentAmount);
    totSpentAmount = totSpentAmount + spentAmount;
}

if(_buybackFee != 0){
    spentAmount = contractTokenBalance.div(totFee).mul(_buybackFee);
    swapTokensForBNB(spentAmount);
    totSpentAmount = totSpentAmount + spentAmount;
}
```

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 setAllFeePercent(uint8 taxFee, uint8 liquidityFee, uint8 burnFee, uint8 walletFee, uint8 buyba
    require(taxFee &gt;= 0 &amp;&amp; taxFee = 0 &amp;&amp; liquidityFee = 0 &amp;&amp; burnFee = 0 &am
    _taxFee = taxFee;
    _liquidityFee = liquidityFee;
    _burnFee = burnFee;
    _buybackFee = buybackFee;
    _walletFee = walletFee;
}
```

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.

Costly operations inside a loop

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

```
function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Already excluded");
    for (uint256 i = 0; i &lt; _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

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 `SSTOREs`, which might lead to an `out-of-gas`.

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

Wrong require statement for unlock function ✅ Acknowledged

```
//Unlocks the contract for owner when _lockTime is exceeds
function unlock() public virtual {
    require(_previousOwner == msg.sender, "You don't have permission to unlock the token contract");
    require(block.timestamp &gt; _lockTime , "Contract is locked until 7 days");
    emit OwnershipTransferred(_owner, _previousOwner);
    _owner = _previousOwner;
}
```

Recommendation

'_lockTime' is a variable entered in the 'lock' function. The require statement compares the timestamp to locktime. However, the require statement always yields "Contract is locked until 7 days". Which, will not always be the case, as 'lockTime' is a variable.

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

No constraints on ‘lockTime’ ✅ Acknowledged

```
//Locks the contract for owner for the amount of time provided
function lock(uint256 time) public virtual onlyOwner {
    _previousOwner = _owner;
    _owner = address(0);
    _lockTime = block.timestamp + time;
    emit OwnershipTransferred(_owner, address(0));
}
```

Recommendation

We recommend to use a require statement to prevent lockTime from exceeding certain limits. Entering a wrong value here will result in losing ownership of the contract for an undesired amount of time. Prevent errors like these from happening by using require statements.

Contract Privileges

Maximum Fee Limit Check

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

Type of fee	Description
Transfer fee	🔴 Owner can set the transfer fee to 25% or higher
Buy fee	🔴 Owner can set the buy fee to 25% or higher
Sell fee	🔴 Owner can set the sell fee to 25% or higher

Note: this is a boolean check to 25%, we will not change this value in the report.

Contract Pausability Check

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	● Owner cannot pause the contract

Max Transaction Amount Check

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Privilege Check	Description
Can owner set max tx amount?	🔴 Owner can set max transaction amount

Exclude From Fees Check

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Privilege Check	Description
Can owner exclude from fees?	● Owner can exclude from fees

Ability To Mint Check

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	🟢 Owner cannot mint new tokens

Ability To Blacklist Check

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	🟢 Owner cannot blacklist addresses

Other Owner Privileges Check

Coinsult lists all important contract methods which the owner can interact with.

✅ No other important owner privileges to mention.

Notes

Notes by Meta Games Coin

No notes provided by the team.

Notes by Coinsult

✅ No notes provided by Coinsult

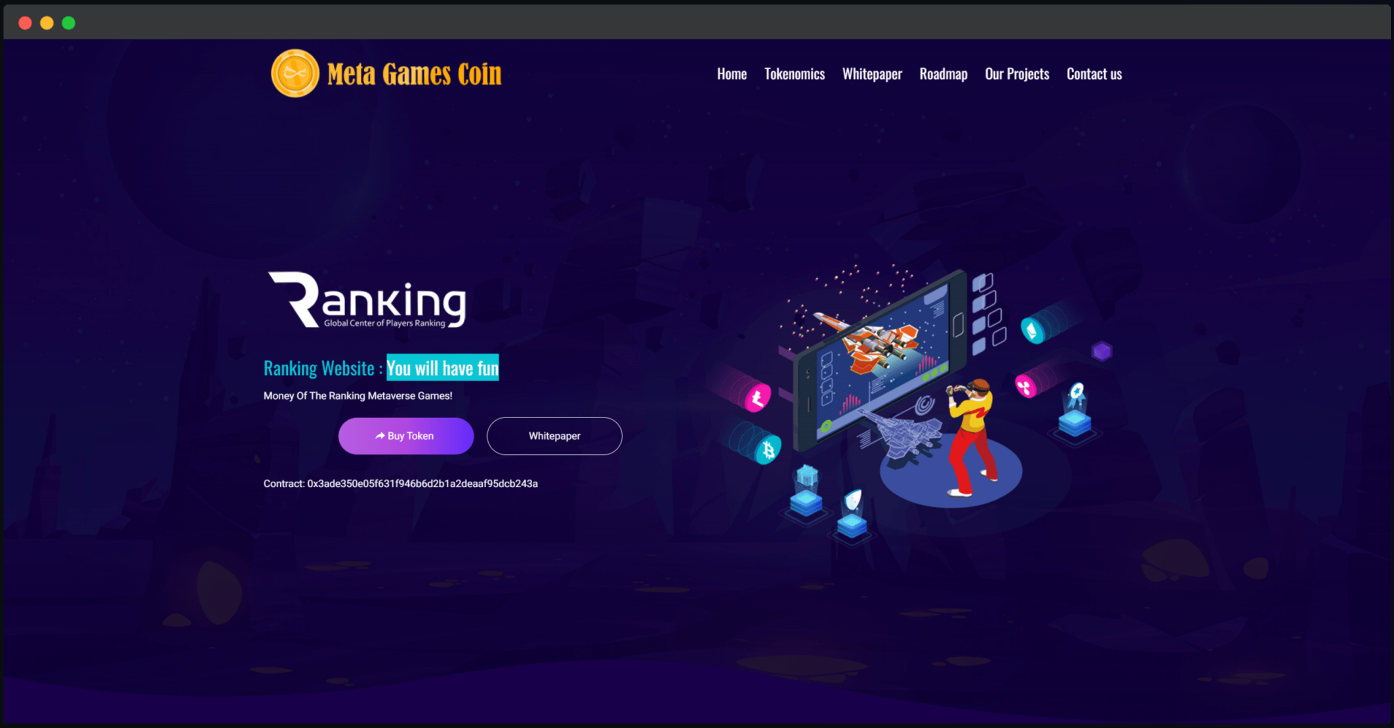
Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract Token is Context, IERC20, Ownable {
using SafeMath for uint256;
using Address for address;
using SafeERC20 for IERC20;

address dead = 0x000000000000000000000000000000000000dEaD;

uint8 public maxLiqFee = 10;
uint8 public maxTaxFee = 10;
uint8 public maxBurnFee = 10;
uint8 public maxWalletFee = 10;
uint8 public maxBuybackFee = 10;
uint8 public minMxTxPercentage = 1;
uint8 public minMxWalletPercentage = 1;

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

mapping (address =&gt; bool) private _isExcludedFromFee;

mapping (address =&gt; bool) private _isExcluded;
address[] private _excluded;

address public router = 0x10ED43C718714eb63d5aA57B78B54704E256024E;
//address public router = 0xD99D1c33F9fC3444f8101754aBC46c52416550D1;

uint256 private constant MAX = ~uint256(0);
uint256 public _tTotal;
uint256 private _rTotal;
uint256 private _tFeeTotal;

bool public mintedByMudra = true;

string public _name;
string public _symbol;
uint8 private _decimals;
```

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.



Ranking Website : You will have fun

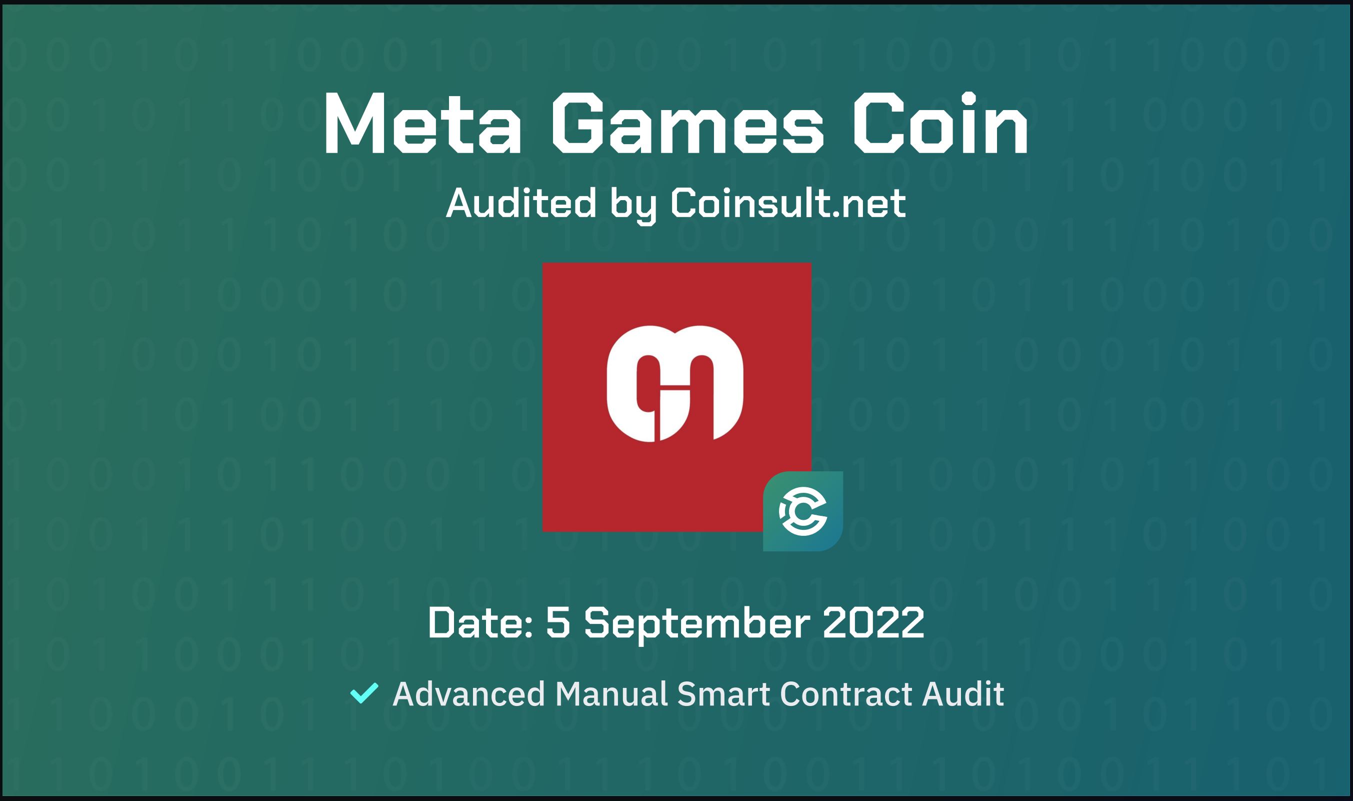
Money Of The Ranking Metaverse Games!

Contract: 0x3ade350e05f631f946b6d2b1a2deaaf95dcb243a

Type of check	Description
Mobile friendly?	● The website is mobile friendly
Contains jQuery errors?	● The website does not contain jQuery errors
Is SSL secured?	● The website is SSL secured
Contains spelling errors?	● The website does not contain spelling errors

Certificate of Proof

● Not KYC verified by Coinsult



Meta Games Coin

Audited by Coinsult.net

Date: 5 September 2022

✓ Advanced Manual Smart Contract Audit

End of report

Smart Contract Audit