Coinsult

Advanced Manual Smart Contract Audit

Project: Bluetherium

Website: https://bluetherium.org/

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

0xef4eC6011C6feD75abccc51aF378A26894865F87

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	0x462a4c11b6ac4ca9483bb1fd66100b4d1f940360	30,000,000	30.0000%
2	0x1a630e28a034c9281075db9d9cf3c22cf892ed14	12,000,000	12.0000%
3	0x9c14a4836f91c23a7601873b1c2a3d7a31beb4a4	10,000,000	10.0000%
4	0xb1307479399af2d7631fe0ba9a38c90d1efb3d34	10,000,000	10.0000%
5	0x1fd86574c5769cff472a288f04b41fa4d43c7dc3	10,000,000	10.0000%
6	0xdb9560edf20377c8521a2bccf5189c888500404d	10,000,000	10.0000%
7	0x3a3ec9561abbfceb6751de7369652a510a9f7d76	10,000,000	10.0000%
8	0x6e6659d86fa94e54ad1e076b7d2d0e295e5c1b10	5,000,000	5.0000%
9	0xc0bba09235c527aa3ad9b93f42f26a20cf2f3855	3,000,000	3.0000%

Source Code

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

https://bscscan.com/address/0xef4eC6011C6feD75abccc51aF378A26894865F87#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.

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 sender,
    address recipient,
    uint256 amount
) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(sender, recipient, amount);

    uint256 senderBalance = _balances[sender];
    require(senderBalance &gt;= amount, "ERC20: transfer amount exceeds balance");
    unchecked {
        _balances[sender] = senderBalance - amount;
    }
    _balances[recipient] += amount;

    emit Transfer(sender, recipient, amount);

    _afterTokenTransfer(sender, recipient, 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.

```
uniswapV2Router02.swapExactTokensForETH(
    toSell,
    0,
    sellPath,
    address(this),
    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.

```
uint256 private swapThreshold = 0.0000005 ether; // The contract will only swap to ETH, once the fee
```

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.

No zero address validation for some functions

Detect missing zero address validation.

```
function setTaxWallets(address dev, address marketing, address charity) public onlyOwner {
    taxWallets["dev"] = dev;
    taxWallets["marketing"] = marketing;
    taxWallets["charity"] = charity;
}
```

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.

Divide before multiply

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

```
uint256 baseUnit = amount / denominator;
if(from == address(uniswapV2Pair)) {
    tax += baseUnit * buyTaxes["marketing"];
    tax += baseUnit * buyTaxes["dev"];
    tax += baseUnit * buyTaxes["liquidity"];
    tax += baseUnit * buyTaxes["charity"];
```

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.

```
/**
 * @dev Sets tax for buys.
 */
function setBuyTax(uint256 dev, uint256 marketing, uint256 liquidity, uint256 charity) public onlyOw
    buyTaxes["dev"] = dev;
    buyTaxes["marketing"] = marketing;
    buyTaxes["liquidity"] = liquidity;
    buyTaxes["charity"] = charity;
}

/**
 * @dev Sets tax for sells.
 */
function setSellTax(uint256 dev, uint256 marketing, uint256 liquidity, uint256 charity) public onlyO

    sellTaxes["dev"] = dev;
    sellTaxes["marketing"] = marketing;
    sellTaxes["liquidity"] = liquidity;
    sellTaxes["charity"] = charity;
```

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.

Owner privileges

- Owner cannot change max transaction amount
- Owner can set fees higher than 25%
- Owner can exclude from fees
- Owner can pause the contract
- Owner can blacklist addresses

Extra notes by the team

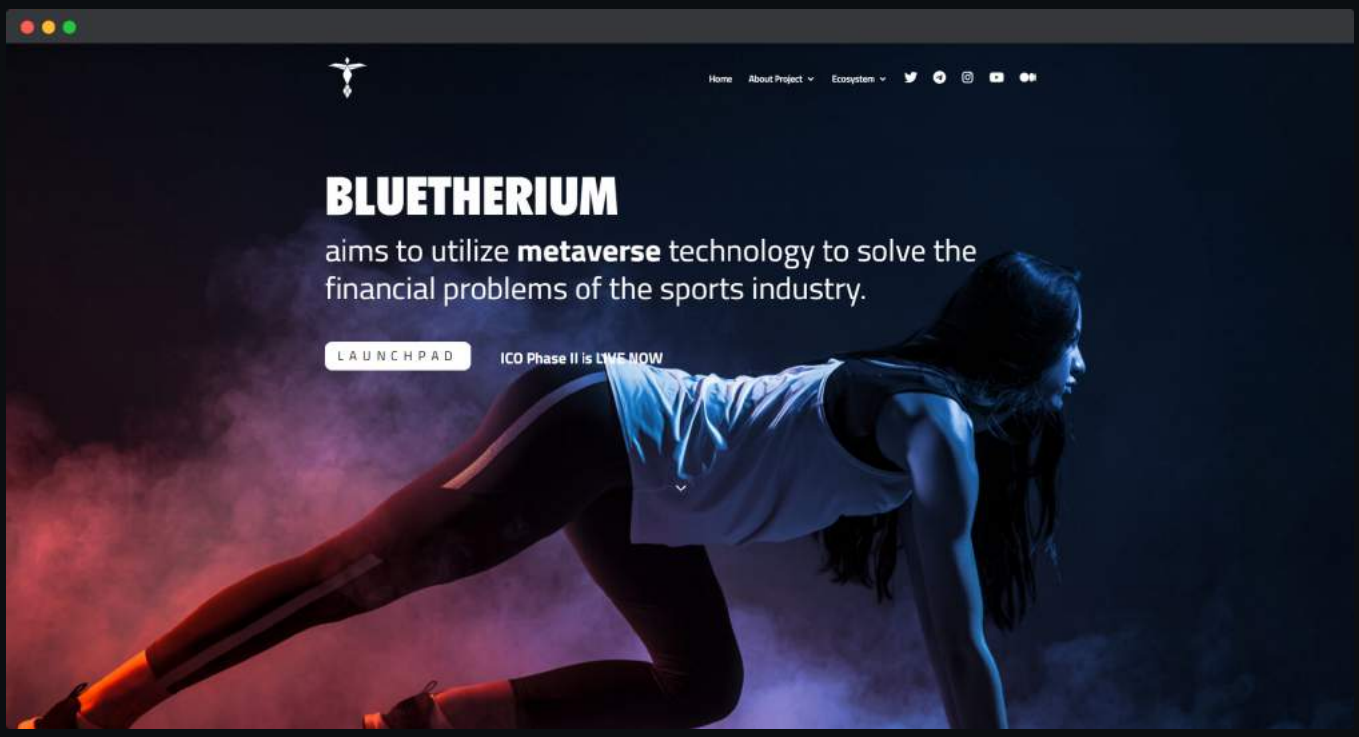
No notes

Contract Snapshot

```
contract CoinToken is ERC20, Ownable, Pausable {

// CONFIG START

uint256 private initialSupply;

uint256 private denominator = 100;

uint256 private swapThreshold = 0.0000005 ether; // The contract will only swap to ETH, once the fee

uint256 private devTaxBuy;
uint256 private marketingTaxBuy;
uint256 private liquidityTaxBuy;
uint256 private charityTaxBuy;

uint256 private devTaxSell;
uint256 private marketingTaxSell;
uint256 private liquidityTaxSell;
uint256 private charityTaxSell;

address private devTaxWallet;
address private marketingTaxWallet;
address private liquidityTaxWallet;
address private charityTaxWallet;

// CONFIG END

mapping (address =&gt; bool) private blacklist;
mapping (address =&gt; bool) private excludeList;

mapping (string =&gt; uint256) private buyTaxes;
mapping (string =&gt; uint256) private sellTaxes;
mapping (string =&gt; address) private taxWallets;

bool public taxStatus = true;

IUniswapV2Router02 private uniswapV2Router02;
```

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

KYC verified by Coinsult