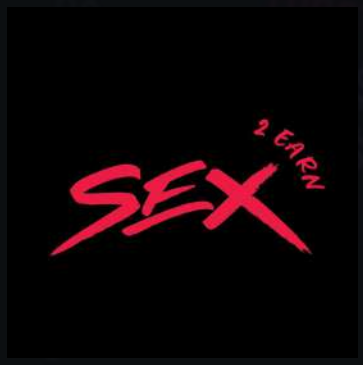
Coinsult

Advanced Manual Smart Contract Audit



Project: Sex To Earn

Website: No website

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

Contract Address

0xe7BD139FC3CD3Ae9F8Cf59f1fC2e966cECceb407

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	0x2a59b7147b8e522894b2fb870cebc776b5f715e0	100,000,000	100.0000%

Source Code

Coinsult was comissioned by Sex To Earn to perform an audit based on the following smart contract:

https://bscscan.com/address/0xe7bd139fc3cd3ae9f8cf59f1fc2e966cecceb407#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

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

    if (amount == 0) {
        super._transfer(from, to, 0);
        return;
    }

    if (!tradingActive) {
        require(
            _isExcludedFromFees[from] || _isExcludedFromFees[to],
            "Trading is not active."
        );
    }

    if (limitsInEffect) {
        if (
```

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.

Too many digits

Literals with many digits are difficult to read and review.

```
newAmount &gt;= (totalSupply() * 1) / 100000,
```

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 updateDevWallet(address newWallet) external onlyOwner {
    emit devWalletUpdated(newWallet, devWallet);
    devWallet = newWallet;
}
```

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.

Functions that send Ether to arbitrary destinations

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

```
function swapBack() private {
        uint256 contractBalance = balanceOf(address(this));

        // prevent extremely large dumps.
        if (contractBalance &gt; swapTokensAtAmount * 5) {
            contractBalance = swapTokensAtAmount * 5;
        }

        uint256 totalTokensToSwap = tokensForLiquidity +
            tokensForMarketing +
            tokensForDev +
            tokensForBuyBack;
        bool success;

        if (contractBalance == 0 || totalTokensToSwap == 0) {
            return;
        }

        // Halve the amount of liquidity tokens
        uint256 liquidityTokens = (contractBalance * tokensForLiquidity) /
            totalTokensToSwap /
            2;
```

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.

Write after write

Variables that are written but never read and written again.

```
function swapBack() private {
        uint256 contractBalance = balanceOf(address(this));

        // prevent extremely large dumps.
        if (contractBalance &gt; swapTokensAtAmount * 5) {
            contractBalance = swapTokensAtAmount * 5;
        }

        uint256 totalTokensToSwap = tokensForLiquidity +
            tokensForMarketing +
            tokensForDev +
            tokensForBuyBack;
        bool success;

        if (contractBalance == 0 || totalTokensToSwap == 0) {
            return;
        }

        // Halve the amount of liquidity tokens
        uint256 liquidityTokens = (contractBalance * tokensForLiquidity) /
            totalTokensToSwap /
            2;
```

Recommendation

Fix or remove the writes.

Exploit scenario

```
```solidity
contract Buggy{
    function my_func() external initializer{
        // ...
        a = b;
        a = c;
        // ..
    }
}
```

`a` is first asigned to `b`, and then to `c`. As a result the first write does nothing.
```

🟢 **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.

```
fees = amount.mul(sellTotalFees).div(100);
```

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 updateBuyFees(
    uint256 _marketingFee,
    uint256 _liquidityFee,
    uint256 _devFee,
    uint256 _buyBackFee
) external onlyOwner {
    buyMarketingFee = _marketingFee;
    buyLiquidityFee = _liquidityFee;
    buyDevFee = _devFee;
    buyBuyBackFee = _buyBackFee;
    buyTotalFees =
        buyMarketingFee +
        buyLiquidityFee +
        buyDevFee +
        buyBuyBackFee;
    require(buyTotalFees &lt;= 20, &quot;Must keep fees at 20% or less&quot;);
}
```

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 calldata) {
    this;
    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.

🟢 **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 swapBack() private {
        uint256 contractBalance = balanceOf(address(this));

        // prevent extremely large dumps.
        if (contractBalance &gt; swapTokensAtAmount * 5) {
            contractBalance = swapTokensAtAmount * 5;
        }

        uint256 totalTokensToSwap = tokensForLiquidity +
            tokensForMarketing +
            tokensForDev +
            tokensForBuyBack;
        bool success;

        if (contractBalance == 0 || totalTokensToSwap == 0) {
            return;
        }
```

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`.

Owner privileges

- Owner can change max transaction amount
- Owner can set fees higher than 25%
- Owner can exclude from fees
- Owner can pause the contract
- ⚠ Owner can set max wallet balance

Extra notes by the team

No notes

Contract Snapshot

```
contract sex2earn is ERC20, Ownable {
using SafeMath for uint256;

IUniswapV2Router02 public immutable uniswapV2Router;
address public immutable uniswapV2Pair;
address public constant deadAddress = address(0xdead);

bool private swapping;

address public marketingWallet;
address public devWallet;
address public buyBackWallet;

uint256 public maxTransactionAmount;
uint256 public swapTokensAtAmount;
uint256 public maxWallet;
```

Project Overview

Not KYC verified by Coinsult