Coinsult

Advanced Manual Smart Contract Audit

Project: Milkyfinance

Website: http://Milkyfinance.io

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

Contract Address

0xA49abB903a84Bcc4663d2f81145254d5Ab2E5BcA

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	0x47472ce42ad63f769ab3a3a88e7bc7767b8e57ab	1,000,000,000	100.0000%

Source Code

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

https://bscscan.com/address/0xA49abB903a84Bcc4663d2f81145254d5Ab2E5BcA#code

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

5 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
) private {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount &gt; 0, "Transfer amount must be greater than zero");

    if(from != owner() &amp;&amp; to != owner() &amp;&amp; to != uniswapV2Pair)
        require(balanceOf(to) + amount = SWAP_TOKENS_AT_AMOUNT &amp;&amp; !swapping &amp;&amp; from
        swapping = true;
        uint256 sellTokens = balanceOf(address(this));
        swapAndSendToFee(sellTokens);
        swapping = false;
    }

    _tOwned[from] -= amount;
    uint256 transferAmount = amount;

    //if any account belongs to _isExcludedFromFee account then remove the fee
    if(! isExcludedFromFee[from] &amp;&amp; ! isExcludedFromFee[to]){
```

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 addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    (,uint256 ethFromLiquidity,) = uniswapV2Router.addLiquidityETH {value: ethAmount} (
        address(this),
        tokenAmount,
        0,
        0,
        owner(),
        block.timestamp
    );

    if (ethAmount - ethFromLiquidity &gt; 0)
        payable(_marketingWallet).sendValue (ethAmount - ethFromLiquidity);
}
```

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 _tTotal = 1000000000 * 10**_decimals;
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function update_Tax(uint256 taxBuy, uint256 taxSell, uint256 taxSell_marketing, uint256 taxSell_dev,
    require(taxBuy&lt;4 &amp;&amp;  taxSell&lt;4 &amp;&amp; taxSell_marketing+taxSell_dev+taxSell_fu
    _taxBuy = taxBuy;
    _taxSell = taxSell;
    _slotMarketing = taxSell_marketing;
    _slotDev = taxSell_dev;
    _slotFund = taxSell_fund;
    _liquidityFee = liquidityFee;
}
```

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/et
    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

Extra notes by the team

No notes

Contract Snapshot

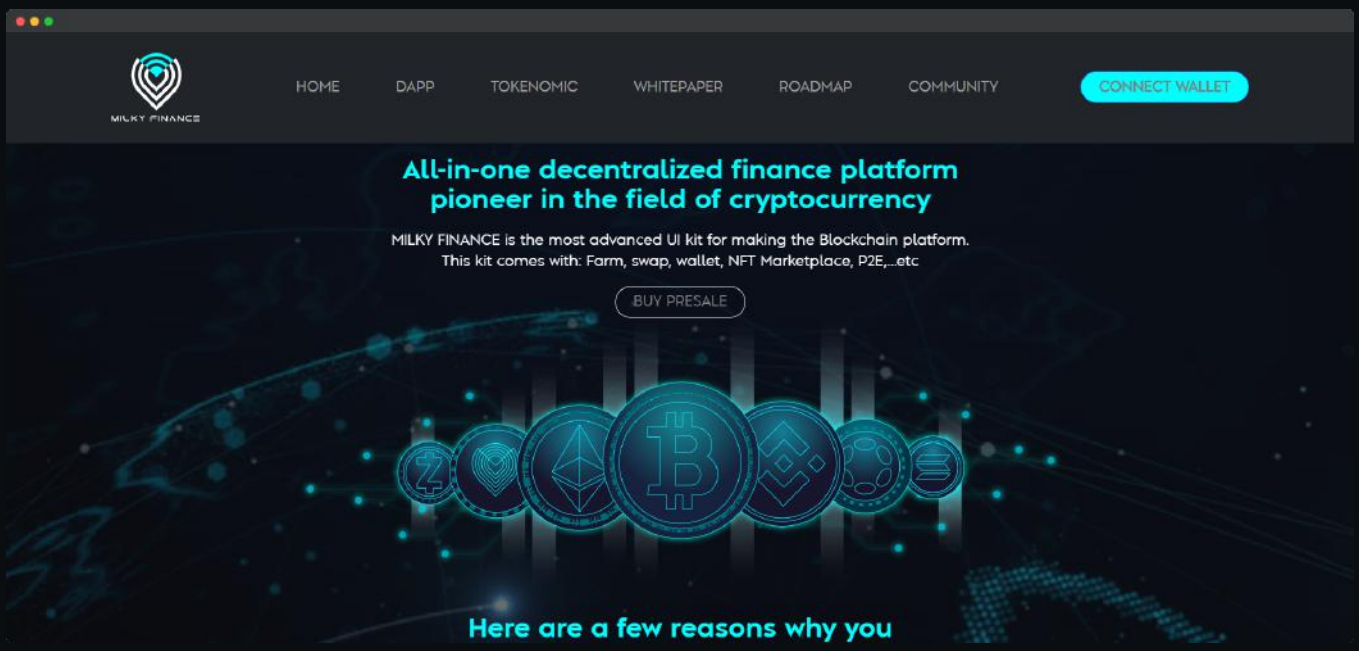
```
contract MilkyFinance is Context, IERC20, Ownable {

string private constant _name = "Milky Finance";
string private constant _symbol = "MIFI";
uint8 private constant _decimals = 18;

uint256 private _tTotal = 1000000000 * 10**_decimals;
uint256 public  _maxTxAmount = 1000000000 * 10**_decimals;           // balance of receiver after bu
uint256 private constant SWAP_TOKENS_AT_AMOUNT = 3 * 10**_decimals; // minimum sto swap token to bnb
uint256 private _liquidityFee;
```

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

Not KYC verified by Coinsult

Milkyfinance

Audited by Coinsult.net

Date: 11 July 2022

Advanced Manual Smart Contract Audit