Coinsult

Advanced Manual Smart Contract Audit

Project: Get schiffy Gold

Website: https://getschiffy.com/en

● Low-Risk	● Medium-Risk	● High-Risk
5 low-risk code issues found	1 medium-risk code issues found	0 high-risk code issues found

Contract Address

0x238eeffc574d93c8bed5e960b2df7ddb5a22d402

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

1	0x79e7e7c33f51a7810c7c6d86cb740ba3aa5f56bf	844,926,696,156.400007542085677827	84.4927%
2	PancakeSwap V2: $GOLD 12	24,152,542,247.356501172361835905	2.4153%
3	0xfe64d9b21bc114e6f1ccaef73ba519b327732713	5,748,867,223.64014	0.5749%
4	0xed073ecbd2ae03633d3c3e9f1b743c5f84ec845e	5,301,017,329.67207647920907125	0.5301%
5	0x6c3db90346dcabf8d8433a2d85346bb3e030481e	4,446,564,287.14258	0.4447%

Source Code

Coinsult was comissioned by Get schiffy Gold to perform an audit based on the following smart contract:

https://bscscan.com/address/0x238eeffc574d93c8bed5e960b2df7ddb5a22d402#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

5 low-risk code issues found

🟡 Medium-Risk

1 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");
    uint256 contractTokenBalance = balanceOf(address(this));
    bool overMinimumTokenBalance = contractTokenBalance &gt;= minimumTokensBeforeSwap;
    if (!inSwapAndLiquify &amp;amp; swapAndLiquifyEnabled &amp;amp; to == uniswapV2Pair) {
        if (overMinimumTokenBalance) {
            contractTokenBalance = minimumTokensBeforeSwap;
            if(swapInSwapToken) {
                swapTokensInSwapToken(contractTokenBalance);
            } else {
                swapTokens(contractTokenBalance);
            }
        }
    }
    bool takeFee = true;
    //if any account belongs to _isExcludedFromFee account then remove the fee
    if(_isExcludedFromFee[from] || _isExcludedFromFee[to] || to != uniswapV2Pair){
        takeFee = false;
    }
    _tokenTransfer(from,to,amount,takeFee);
```

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 unlock() public virtual {
    require(_previousOwner == msg.sender, "You don't have permission to unlock");
    require(block.timestamp &gt; _lockTime , "Contract is locked until 7 days");
    emit OwnershipTransferred(_owner, _previousOwner);
    _owner = _previousOwner;
}
```

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.

No zero address validation for some functions

Detect missing zero address validation.

```
function setSwapToken(address _swapToken) external onlyOwner {
    swapTokenAddress = _swapToken;
    emit SwapTokenChanged(_swapToken);
}
```

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.

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.

● **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], "Account is 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;
        }
    }
    emit IncludedInReward(account);
}
```

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`.

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

Potential require statement issue

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

Recommendation

The second require statement can return a wrong value when false. _LockTime can be changed by using the lock() function so the return value “Contract is locked until 7 days” may be wrong when function lock() is changed to a different value than 7 days.

Set a correct return value for the second require statement.

Owner privileges

- 🟢 Owner cannot set fees higher than 25%
- 🟢 Owner cannot pause trading
- 🟢 Owner cannot change max transaction amount
- 🟡 Owner can exclude from fees
- ⚠️ Owner can exclude addresses from reward
- ⚠️ Owner can lock the contract for unlimited amount of time
- ⚠️ Owner can change minimum amount of before swap

Extra notes by the team

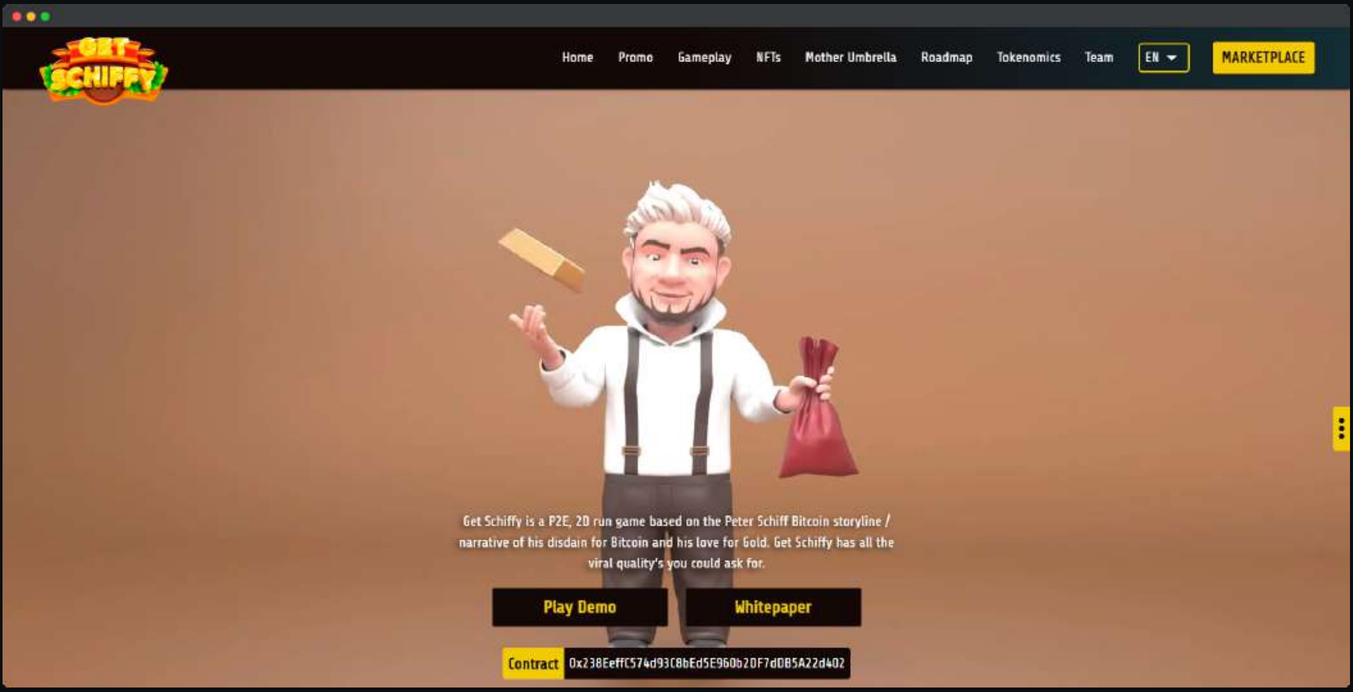
No notes

Contract Snapshot

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

address payable public marketingAddress; // Marketing Address
address public swapTokenAddress = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56; // swap token default
address public immutable deadAddress = 0x000000000000000000000000000000000000dEaD;
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;
```

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.



Get Schiffy is a P2E, 2D run game based on the Peter Schiff Bitcoin storyline / narrative of his disdain for Bitcoin and his love for Gold. Get Schiffy has all the viral quality's you could ask for.

Contract 0x238EeffC574d93C8bEd5E960b20F7dDB5A22d402

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

Project Overview

● Not KYC verified by Coinsult

Get schiffy Gold

Audited by Coinsult.net

Date: 22 July 2022

✓ Advanced Manual Smart Contract Audit