Coinsult

Advanced Manual Smart Contract Audit

Project: Crypney

Website: https://crypney.com

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

Contract Address

0xF4c396aA6FB0321Df4F8bAF9Cf6f433be6ae342c

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	0x2a135152ffb5b704c9a546449f2b6da7016f763b	100,000,000	100.0000%

Source Code

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

https://bscscan.com/address/0xf4c396aa6fb0321df4f8baf9cf6f433be6ae342c#code

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

7 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");
        require(!_isBlacklisted[from] &amp;&amp; !_isBlacklisted[to], 'Blacklisted address');

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

uint256 contractTokenBalance = balanceOf(address(this));

        bool canSwap = contractTokenBalance &gt;= swapTokensAtAmount;

        if( canSwap &amp;&amp;
            !swapping &amp;&amp;
            !automatedMarketMakerPairs[from] &amp;&amp;
            from != owner() &amp;&amp;
```

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 canAutoClaim(uint256 lastClaimTime) private view returns (bool) {
    if(lastClaimTime &gt; block.timestamp)  {
        return false;
    }

    return block.timestamp.sub(lastClaimTime) &gt;= claimWait;
}
```

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.

```
function updateGasForProcessing(uint256 newValue) public onlyOwner {
    require(newValue &gt;= 200000 &amp;&amp; newValue &lt;= 500000, &quot;card: gasForProcessing mus
    require(newValue != gasForProcessing, &quot;card: Cannot update gasForProcessing to same value&q
    emit GasForProcessingUpdated(newValue, gasForProcessing);
    gasForProcessing = newValue;
}
```

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 setMarketingWallet(address payable wallet) external onlyOwner{
    _marketingWalletAddress = wallet;
}
```

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 addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {

        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            address(0),
            block.timestamp
        );
```

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.

Unchecked transfer

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

```
function swapAndSendToFee(uint256 tokens) private  {

    uint256 initialBUSDBalance = IERC20(BUSD).balanceOf(address(this));

    swapTokensForBUSD(tokens);
    uint256 newBalance = (IERC20(BUSD).balanceOf(address(this))).sub(initialBUSDBalance);
    IERC20(BUSD).transfer(_marketingWalletAddress, newBalance);
}
```

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.

Redundant Statements

Detect the usage of redundant statements that have no effect.

```
function _msgData() internal view virtual returns (bytes calldata) {
    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
- 🔴 Owner can blacklist addresses
- ⚠️ Owner can exclude addresses from dividend
- ⚠️ Owner can update claimwait

Extra notes by the team

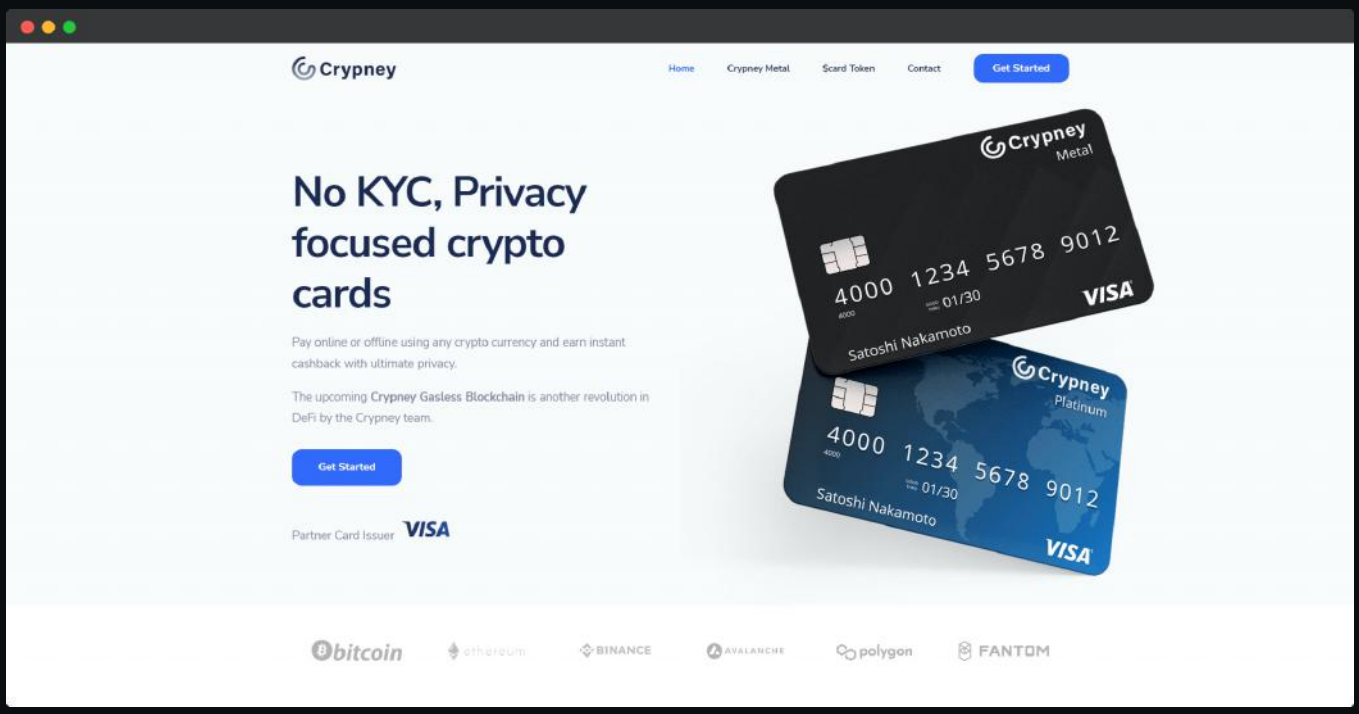
No notes

Contract Snapshot

```
constructor() public ERC20("Crypney - Crypto Credit Card", "card") {

        dividendTracker = new cardDividendTracker();


        IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E
         // Create a uniswap pair for this new token
        address _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this), _uniswapV2Router.WETH());

        uniswapV2Router = _uniswapV2Router;
        uniswapV2Pair = _uniswapV2Pair;

        _setAutomatedMarketMakerPair(_uniswapV2Pair, true);
```

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