Coinsult

Advanced Manual Smart Contract Audit

Project: Fuel

Website: http://fueltoken.net/

🟢 **Low-Risk**

6 low-risk code
issues found

🟡 **Medium-Risk**

0 medium-risk code
issues found

🔴 **High-Risk**

0 high-risk code
issues found

Contract Address

0x689500593721f3202Ee994d11007741959fad183

Disclaimer

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x50833d3390d947b86a0a59a1bf222b5129ee3311	100,000,000	100.0000%

Source Code

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

https://bscscan.com/address/0x689500593721f3202Ee994d11007741959fad183#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 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())
        require(amount = _maxTxAmount)
    {
        contractTokenBalance = _maxTxAmount;
    }

    bool overMinTokenBalance = contractTokenBalance &gt;= numTokensSellToAddToLiquidity;
    if (
        overMinTokenBalance &amp;&amp;
        !inSwapAndLiquify &amp;&amp;
        from != uniswapV2Pair &amp;&amp;
        swapAndLiquifyEnabled
    ) {
        contractTokenBalance = numTokensSellToAddToLiquidity;
```

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.

```solidity
uint256 private _tTotal = 100000000 * 10**18;
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```solidity
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.

Unchecked transfer

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

```
function removeStuckToken(address _address) external onlyOwner {
    require(_address != address(this), "Can't withdraw tokens destined for liquidity");
    require(IERC20(_address).balanceOf(address(this)) &gt; 0, "Can't withdraw 0");

    IERC20(_address).transfer(owner(), IERC20(_address).balanceOf(address(this)));
}
```

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..

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setFeePercent(uint256 taxFee, uint256 liquidityFee, uint256 marketingFee) external onlyOwner
    _taxFee = taxFee;
    _liquidityFee = liquidityFee;
    _marketingFee = marketingFee;
    require(_taxFee + _liquidityFee + _marketingFee <= 20,"Taxes can't be set more then
}
```

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.

```
Address._functionCallWithValue(address,bytes,uint256,string) (#357-378) is never used and should be
Address.functionCall(address,bytes) (#317-319) is never used and should be removed
Address.functionCall(address,bytes,string) (#327-329) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (#342-344) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (#352-355) is never used and should be r
Address.isContract(address) (#264-273) is never used and should be removed
Address.sendValue(address,uint256) (#291-297) is never used and should be removed
Context._msgData() (#236-239) is never used and should be removed
SafeMath.mod(uint256,uint256) (#209-211) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (#225-228) is never used and should be removed
```

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.

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 < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

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 cannot set fees higher than 25%

- 🟢 Owner cannot pause trading

- 🟡 Owner can change max transaction amount

- 🟡 Owner can exclude from fees

Extra notes by the team

No notes

Contract Snapshot

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

mapping (address => uint256) private _rOwned;
mapping (address => uint256) private _tOwned;
mapping (address => mapping (address => uint256)) private _allowances;

mapping (address => bool) private _isExcludedFromFee;

mapping (address => bool) private _isExcluded;
address[] private _excluded;

uint256 private constant MAX = ~uint256(0);
uint256 private _tTotal = 100000000 * 10**18;
uint256 private _rTotal = (MAX - (MAX % _tTotal));
uint256 private _tFeeTotal;

string private _name = "FUEL";
string private _symbol = "FUEL";
uint8 private _decimals = 18;

uint256 public _marketingFee = 3;
uint256 private _previousMarketingFee = _marketingFee;

uint256 public _taxFee = 3;
uint256 private _previousTaxFee = _taxFee;

uint256 public _liquidityFee = 3;
uint256 private _previousLiquidityFee = _liquidityFee;

address public marketingWallet = 0x2a9C2Ef5D70Ff57A3c44E28EA3003944D9d779b3;

IUniswapV2Router02 public immutable uniswapV2Router;
address public immutable uniswapV2Pair;

bool inSwapAndLiquify;
```
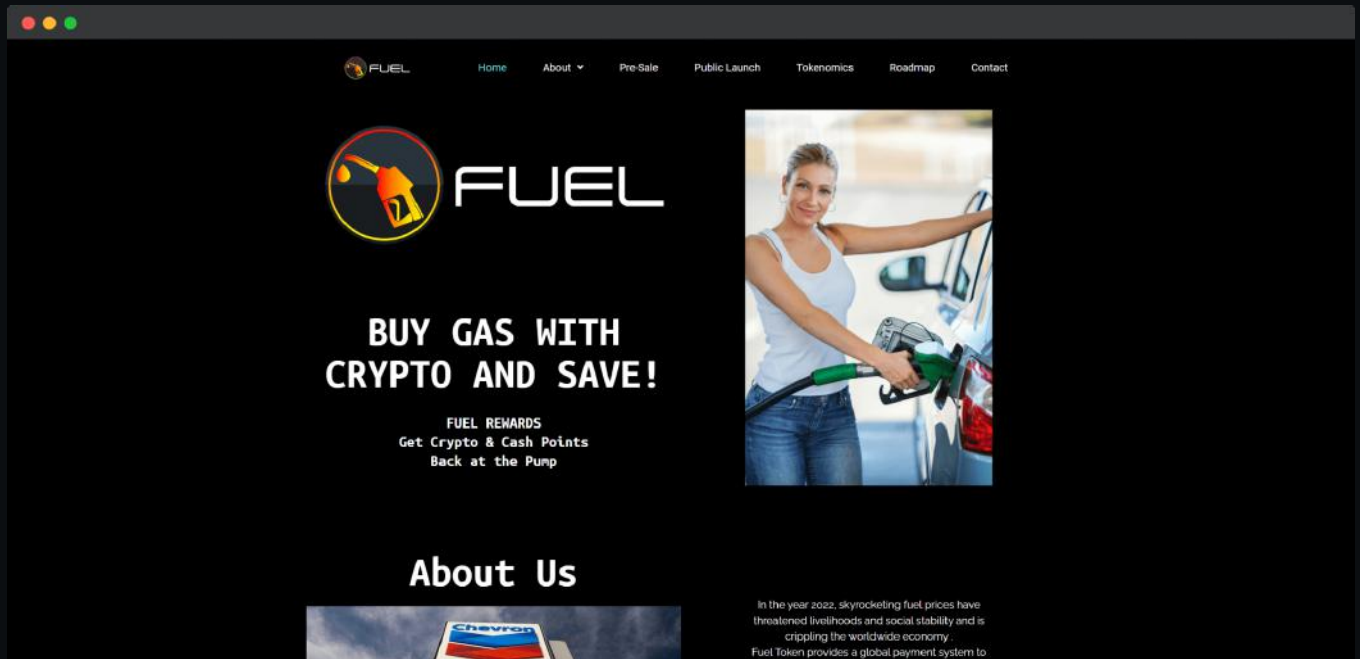
Website Review

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

● Not SSL Secured

● No major spelling errors

Project Overview

🟡 Not KYC verified by Coinsult

Fuel

Audited by Coinsult.net

Date: 16 July 2022

✔ Advanced Manual Smart Contract Audit