Coinsult

Advanced Manual Smart Contract Audit

Project: Web3 Capital

Website: http://web3capital.financial/

● **Low-Risk**	● **Medium-Risk**	● **High-Risk**
8 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

0xdbf23b67027c1a5fd242fd1d61b2846543459854

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	0x2779f1b8daf02d94ddda4fdbc90fc7dea00305c0	1,000,000,000,000,000	100.0000%

Source Code

Coinsult was comissioned by Web3 Capital to perform an audit based on the following smart contract:

https://bscscan.com/address/0xdbf23b67027c1a5fd242fd1d61b2846543459854#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

8 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 returns (bool) {
    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");
    bool buy = false;
    bool sell = false;
    bool other = false;
    if (lpPairs[from]) {
        buy = true;
    } else if (lpPairs[to]) {
        sell = true;
    } else {
        other = true;
    }
    if(_hasLimits(from, to)) {
        if(!tradingEnabled) {
            revert("Trading not yet enabled!");
        }
        if(buy || sell){
            if (!_isExcludedFromLimits[from] &amp;&amp; !_isExcludedFromLimits[to]) {
                require(amount &lt;= _maxTxAmount, &quot;Transfer amount exceeds the maxTxAmount.&qu
            }
```

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.

```
if (timeSinceLastPair != 0) {
    require(block.timestamp - timeSinceLastPair &gt; 3 days, "Cannot set a new pair this week!");
}
```

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 reflectorGas = 300000;
```

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 setWallets(address payable marketing) external onlyOwner {
    _taxWallets.marketing = payable(marketing);
}
```

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.

```
if(ratios.marketing &gt; 0){
    (success,) = _taxWallets.marketing.call{value: marketingBalance, gas: 30000}("");
}
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setWallets(address payable marketing) external onlyOwner {
    _taxWallets.marketing = payable(marketing);
}

function setMaxTxPercent(uint256 percent, uint256 divisor) external onlyOwner {
    require((_tTotal * percent) / divisor &gt;= (_tTotal / 1000), "Max Transaction amt must be above
    _maxTxAmount = (_tTotal * percent) / divisor;
}
```

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.

Boolean equality

Detects the comparison to boolean constants.

```
function setLpPair(address pair, bool enabled) external onlyOwner {
    if (enabled == false) {
        lpPairs[pair] = false;
        antiSnipe.setLpPair(pair, false);
    } else {
        if (timeSinceLastPair != 0) {
            require(block.timestamp - timeSinceLastPair &gt; 3 days, "Cannot set a new pair this weel
        }
        lpPairs[pair] = true;
        timeSinceLastPair = block.timestamp;
        antiSnipe.setLpPair(pair, true);
    }
}
```

Recommendation

Remove the equality to the boolean constant.

Exploit scenario

```
contract A {
    function f(bool x) public {
        // ...
        if (x == true) { // bad!
           // ...
        }
        // ...
    }
}
```

Boolean constants can be used directly and do not need to be compare to `true` or `false`.

● **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 multiSendTokens(address[] memory accounts, uint256[] memory amounts) external onlyOwner {
    require(accounts.length == amounts.length, "Lengths do not match.");
    for (uint8 i = 0; i = amounts[i]);
        _finalizeTransfer(msg.sender, accounts[i], amounts[i]*10**_decimals, false, false, false, tr
    }
}
```

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 can change max transaction amount
- 🟡 Owner can exclude from fees
- 🟡 Owner can pause the contract
- 🔴 Owner can blacklist addresses

⚠️ Owner can set max wallet size

⚠️ Owner can set max price impact

Extra notes by the team

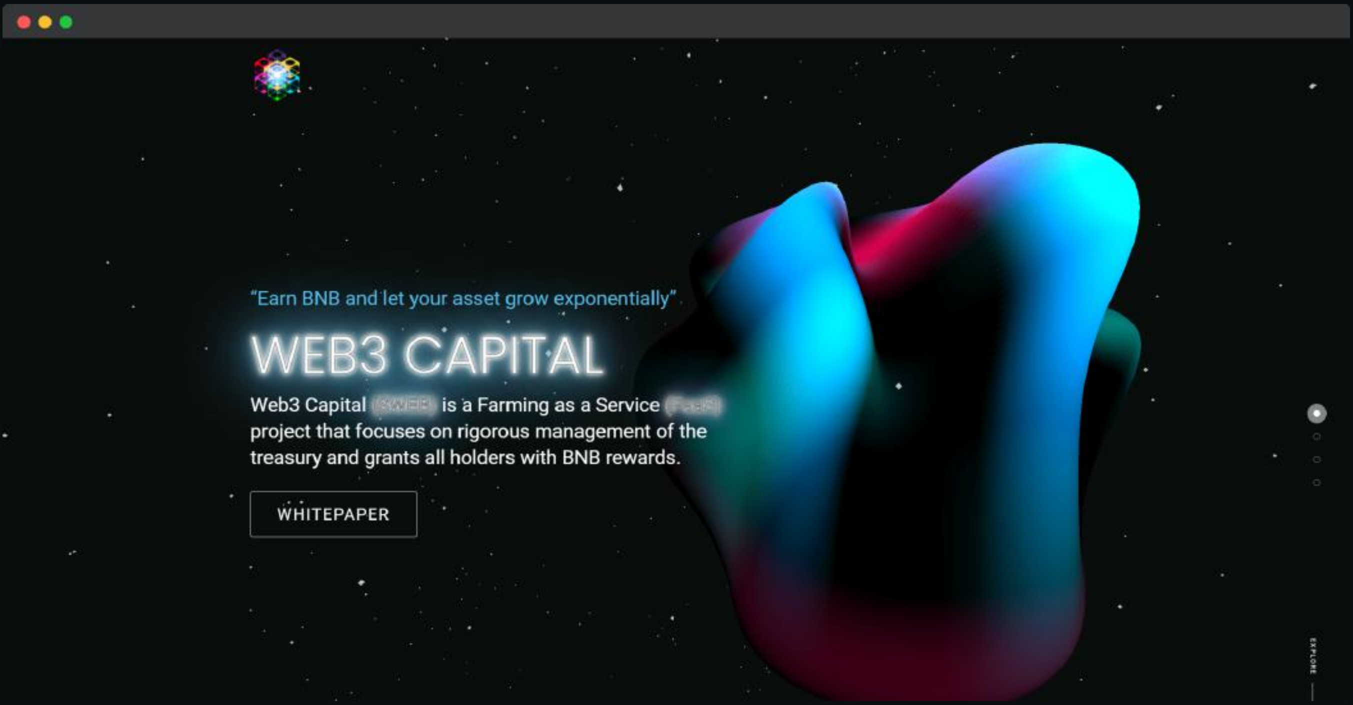
No notes

Contract Snapshot

```
contract Web3Capital is IERC20 {
// Ownership moved to in-contract for customizability.
address private _owner;

mapping (address =&gt; uint256) _tOwned;
mapping (address =&gt; bool) lpPairs;
uint256 private timeSinceLastPair = 0;
mapping (address =&gt; mapping (address =&gt; uint256)) _allowances;
mapping (address =&gt; bool) private _isExcludedFromProtection;
mapping (address =&gt; bool) private _isExcludedFromFees;
mapping (address =&gt; bool) private _isExcludedFromLimits;
mapping (address =&gt; bool) private _isExcludedFromDividends;
mapping (address =&gt; bool) private _liquidityHolders;

mapping (address =&gt; bool) private presaleAddresses;
bool private allowedPresaleExclusion = true;

uint256 constant private startingSupply = 1_000_000_000_000_000;

string constant private _name = "Web3 Capital";
string constant private _symbol = "WEB";
uint8 constant private _decimals = 9;

uint256 constant private _tTotal = startingSupply * (10 ** _decimals);

struct Fees {
    uint16 buyFee;
    uint16 sellFee;
    uint16 transferFee;
}

struct Ratios {
    uint16 rewards;
    uint16 liquidity;
    uint16 marketing;
    uint16 burn;
    uint16 total;
```

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.



"Earn BNB and let your asset grow exponentially"

WEB3 CAPITAL

Web3 Capital [redacted] is a Farming as a Service [redacted] project that focuses on rigorous management of the treasury and grants all holders with BNB rewards.

- 🟢 Mobile Friendly
- 🟢 Does not contain jQuery errors
- 🔴 Not SSL Secured
- 🟢 No major spelling errors

Project Overview

● Not KYC verified by Coinsult

Web3 Capital

Audited by Coinsult.net

WEB3 CAPITAL

Date: 15 July 2022

✓ Advanced Manual Smart Contract Audit