



Coinsult

Advanced Manual Smart Contract Audit




Project: Vaultarium

Website: <https://vaultarium.finance/>

 **Low-Risk**

10 low-risk code
issues found

 **Medium-Risk**

1 medium-risk code
issues found

 **High-Risk**

0 high-risk code
issues found

Contract Address

0x3B7bF09fEe4DC6EFab950F6426fCCA8937c101A0

Disclaimer: Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

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	0xa92f95ffe5dda358c98be00475a638ed357992c6	10,000,000,000	100.0000%

Source Code

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

<https://bscscan.com/address/0x3b7bf09fee4dc6efab950f6426fcca8937c101a0#code>

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

10 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 _transferFrom(address sender, address recipient, uint256 amount) internal returns (bool) {
    if(inSwap){ return _basicTransfer(sender, recipient, amount); }

    checkTxLimit(sender, amount);
    //
    if(shouldSwapBack()){ swapBack(); }
    if(shouldAutoBuyback()){ triggerAutoBuyback(); }

    //      if(!launched() && recipient == pair){ require(_balances[sender] > 0); launch
    _balances[sender] = _balances[sender].sub(amount, "Insufficient Balance");

    uint256 amountReceived = shouldTakeFee(sender) ? takeFee(sender, recipient, amount) : amount;

    _balances[recipient] = _balances[recipient].add(amountReceived);

    if(!isDividendExempt[sender]){ try distributor.setShare(sender, _balances[sender]) {} catch {} }
    if(!isDividendExempt[recipient]){ try distributor.setShare(recipient, _balances[recipient]) {} c

    try distributor.process(distributorGas) {} catch {}

    emit Transfer(sender, recipient, amountReceived);
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if msg.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.

```
router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: msg.value}(
    0,
    path,
    address(this),
    block.timestamp
);
```

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 distributorGas = 500000;
```

Recommendation

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

Exploit scenario

```
contract MyContract{
    uint 1_ether = 1000000000000000000;
}
```

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 setFeeReceivers(address _autoLiquidityReceiver, address _marketingFeeReceiver) external autoLiquidityReceiver = _autoLiquidityReceiver;
marketingFeeReceiver = _marketingFeeReceiver;
}
/**
 * Transfer ownership to new address. Caller must be owner. Leaves old owner authorized
 */
function transferOwnership(address payable adr) public onlyOwner {
    owner = adr;
    authorizations[adr] = true;
    emit OwnershipTransferred(adr);
}
```

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, so Bob 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 swapBack() internal swapping {
    uint256 dynamicLiquidityFee = isOverLiquified(targetLiquidity, targetLiquidityDenominator) ? 0 :
    uint256 amountToLiquify = swapThreshold.mul(dynamicLiquidityFee).div(totalFee).div(2);
    uint256 amountToSwap = swapThreshold.sub(amountToLiquify);

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = WBNB;
    uint256 balanceBefore = address(this).balance;

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        address(this),
        block.timestamp
    );

    uint256 amountBNB = address(this).balance.sub(balanceBefore);

    uint256 totalBNBFee = totalFee.sub(dynamicLiquidityFee.div(2));
```

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 distributeDividend(address shareholder) internal {
    if(shares[shareholder].amount == 0){ return; }

    uint256 amount = getUnpaidEarnings(shareholder);
    if(amount > 0){
        totalDistributed = totalDistributed.add(amount);
        BUSD.transfer(shareholder, amount);
        shareholderClaims[shareholder] = block.timestamp;
        shares[shareholder].totalRealised = shares[shareholder].totalRealised.add(amount);
        shares[shareholder].totalExcluded = getCumulativeDividends(shares[shareholder].amount);
    }
}
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setFees(uint256 _liquidityFee, uint256 _buybackFee, uint256 _reflectionFee, uint256 _marketingFee) {
    liquidityFee = _liquidityFee;
    buybackFee = _buybackFee;
    reflectionFee = _reflectionFee;
    marketingFee = _marketingFee;
    totalFee = _liquidityFee.add(_buybackFee).add(_reflectionFee).add(_marketingFee);
    feeDenominator = _feeDenominator;
    require(totalFee < feeDenominator/4);
}
```

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.

```
modifier onlyBuybacker() { require(buyBacker[msg.sender] == 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.

Conformance to Solidity naming conventions

Allow `_` at the beginning of the `mixed_case` match for private variables and unused parameters.

```
Variable DividendDistributor.BUSD (#240) is not in mixedCase
```

```
Variable DividendDistributor.WBNB (#241) is not in mixedCase
```

Recommendation

Follow the Solidity naming convention.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (ERC20).
- Allow `_` at the beginning of the `mixed_case` match for private variables and unused parameters.

● **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.

```
while(gasUsed < gas && iterations = shareholderCount){
    currentIndex = 0;

    if(shouldDistribute(shareholders[currentIndex])){
        distributeDividend(shareholders[currentIndex]);
    }

    gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
    gasLeft = gasleft();
    currentIndex++;
    iterations++;
}
```

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 `SSTOREs`, which might lead to an out-of-gas.

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

Feedenominator passed through as a parameter

```
function setFees(uint256 _liquidityFee, uint256 _buybackFee, uint256 _reflectionFee, uint256 _marketingFee, uint256 _feeDenominator) {
    liquidityFee = _liquidityFee;
    buybackFee = _buybackFee;
    reflectionFee = _reflectionFee;
    marketingFee = _marketingFee;
    totalFee = _liquidityFee.add(_buybackFee).add(_reflectionFee).add(_marketingFee);
    feeDenominator = _feeDenominator;
    require(totalFee <= feeDenominator/4);
}
```

Recommendation

Hardcode feeDenominator so the total fees can't exceed 25%

Owner privileges

- Owner cannot pause trading
- Owner can change max transaction amount
- Owner can set fees higher than 25%
- Owner can exclude from fees

Extra notes by the team

No notes

Contract Snapshot

```

contract Vaultarium is IBEP20, Auth{
using SafeMath for uint256;

uint256 public constant MASK = type(uint128).max;
address BUSD = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56;
address public WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c;
address DEAD = 0x00000000000000000000000000000000dEaD;
address ZERO = 0x0000000000000000000000000000000000000000;
address DEAD_NON_CHECKSUM = 0x00000000000000000000000000000000dEaD;

string constant _name = "Vaultarium";
string constant _symbol = "VLF";
uint8 constant _decimals = 9;

uint256 _totalSupply = 10_000_000_000 * (10 ** _decimals);
uint256 public _maxTxAmount = _totalSupply.div(400); // 0.25%

mapping (address => uint256) _balances;
mapping (address => mapping (address => uint256)) _allowances;

mapping (address => bool) isFeeExempt;
mapping (address => bool) isTxLimitExempt;
mapping (address => bool) isDividendExempt;

uint256 liquidityFee = 100;
uint256 buybackFee = 100;
uint256 reflectionFee = 400;
uint256 marketingFee = 600;
uint256 totalFee = 1200;
uint256 feeDenominator = 10000;

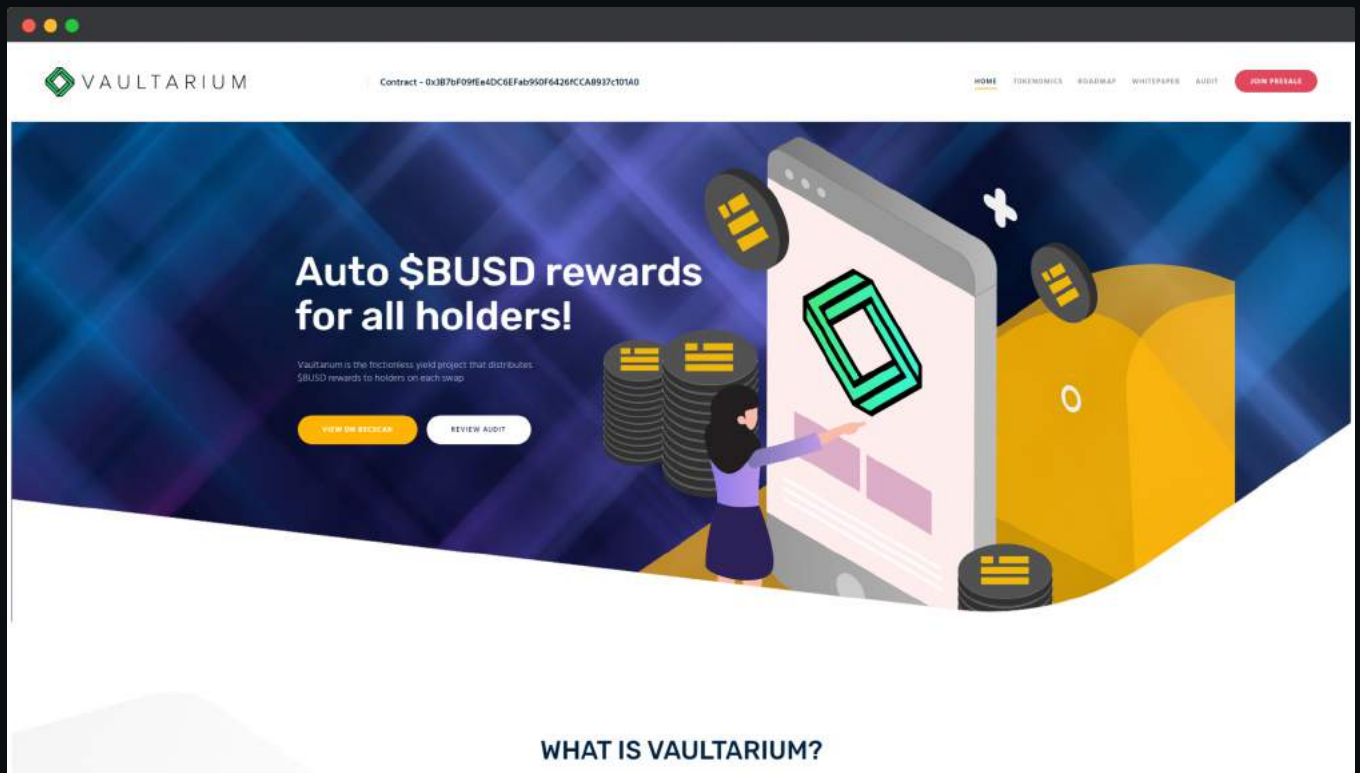
address public autoLiquidityReceiver;
address public marketingFeeReceiver;

uint256 targetLiquidity = 25;
uint256 targetLiquidityDenominator = 100;

```


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

AUDITED
BY COINSULT.NET

