Coinsult

Advanced Manual Smart Contract Audit

Project: RunEarner

Website: https://www.Runearner.com

🟢 Low-Risk	🟡 Medium-Risk	🔴 High-Risk
10 low-risk code issues found	1 medium-risk code issues found	0 high-risk code issues found

Contract Address

0x5B13175262335022EE37c647751DdE2A46b75135

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	0x7cf0e9c4eba47059a942e4f07efb1b94b8d8286d	500,000	100.0000%

Source Code

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

https://bscscan.com/address/0x5B13175262335022EE37c647751DdE2A46b75135#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) {

    require(!blacklist[sender] &amp;&amp; !blacklist[recipient], "in_blacklist");

    if (inSwap) {
        return _basicTransfer(sender, recipient, amount);
    }
    if (shouldRebase()) {
       rebase();
    }

    if (shouldAddLiquidity()) {
        addLiquidity();
    }

    if (shouldSwapBack()) {
        swapBack();
    }
```

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 shouldDistribute(address shareholder) internal view returns (bool) {
    return shareholderClaims[shareholder] + minPeriod  minDistribution;
}
```

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 = 2500000;
```

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 setFeeReceivers(
    address _autoLiquidityReceiver,
    address _treasuryReceiver,
    address _insuranceFundReceiver
) external onlyOwner {
    autoLiquidityReceiver = _autoLiquidityReceiver;
    treasuryReceiver = _treasuryReceiver;
    insuranceFundReceiver = _insuranceFundReceiver;
}
```

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 swapBack() internal swapping {

    uint256 amountToSwap = _gonBalances[address(this)].div(_gonsPerFragment);

    if( amountToSwap == 0) {
        return;
    }

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


    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        address(this),
        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 distributeDividend(address shareholder) internal {
    if(shares[shareholder].amount == 0){ return; }

    uint256 amount = getUnpaidEarnings(shareholder);
    if(amount &gt; 0){
        totalDistributed = totalDistributed.add(amount);
        uint256 reward = amount.div(2);
        uint256 otherreward = amount.sub(reward);
        GMT.transfer(shareholder, reward);
        GMT.approve(address(0x10ED43C718714eb63d5aA57B78B54704E256024E), otherreward);

        address[] memory path = new address[](2);
        path[0] = address(GMT);
        path[1] = router.WETH();
        router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            otherreward,
            0,
            path,
            shareholder,
            block.timestamp
        );
```

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.

Write after write

Variables that are written but never read and written again.

```
function swapBack() internal swapping {

    uint256 amountToSwap = _gonBalances[address(this)].div(_gonsPerFragment);

    if( amountToSwap == 0) {
        return;
    }

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


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

Recommendation

Fix or remove the writes.

Exploit scenario

~~~
```solidity
contract Buggy{
    function my_func() external initializer{
        // ...
        a = b;
        a = c;
        // ..
    }
}
~~~

`a` is first asigned to `b`, and then to `c`. As a result the first write does nothing.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
_gonBalances[address(this)] = _gonBalances[address(this)].add(
            gonAmount.div(feeDenominator).mul(_treasuryFee.add(gmtWbnbFee).add(insuranceFundFee))
```

Recommendation

Consider ordering multiplication before division.

Exploit scenario

```
contract A {
    function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5; n = 10, interest = 2`, coins will be zero. If `(oldSupply * interest / n)` was used, `coins` would have been `1`. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setDistributionCriteria(uint256 _minPeriod, uint256 _minDistribution) external override onl
    minPeriod = _minPeriod;
    minDistribution = _minDistribution;
}
```

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.

Costly operations inside a loop

Costly operations inside a loop might waste gas, so optimizations are justified.

```
function process(uint256 gas) external override onlyToken {
    uint256 shareholderCount = shareholders.length;

    if(shareholderCount == 0) { return; }

    uint256 gasUsed = 0;
    uint256 gasLeft = gasleft();
    uint256 iterations = 0;

    while(gasUsed &lt; gas &amp;&amp; iterations = shareholderCount){
            currentIndex = 0;
        }

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

        gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
```

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.

Unused parameter

```
function takeFee(
    address sender,
    address recipient,
    uint256 gonAmount
) internal  returns (uint256) {
    uint256 _totalFee = totalFee;
    uint256 _treasuryFee = treasuryFee;

    // if (recipient == pair) {
    //       totalFee = totalFee.add(sellFee);
```

Recommendation

Remove unused parameter (recipient) from function input, code is commented.

Owner privileges

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

⚠️ Can blacklist contract addresses
⚠️ Owner can exlude addresses from dividends

Extra notes by the team

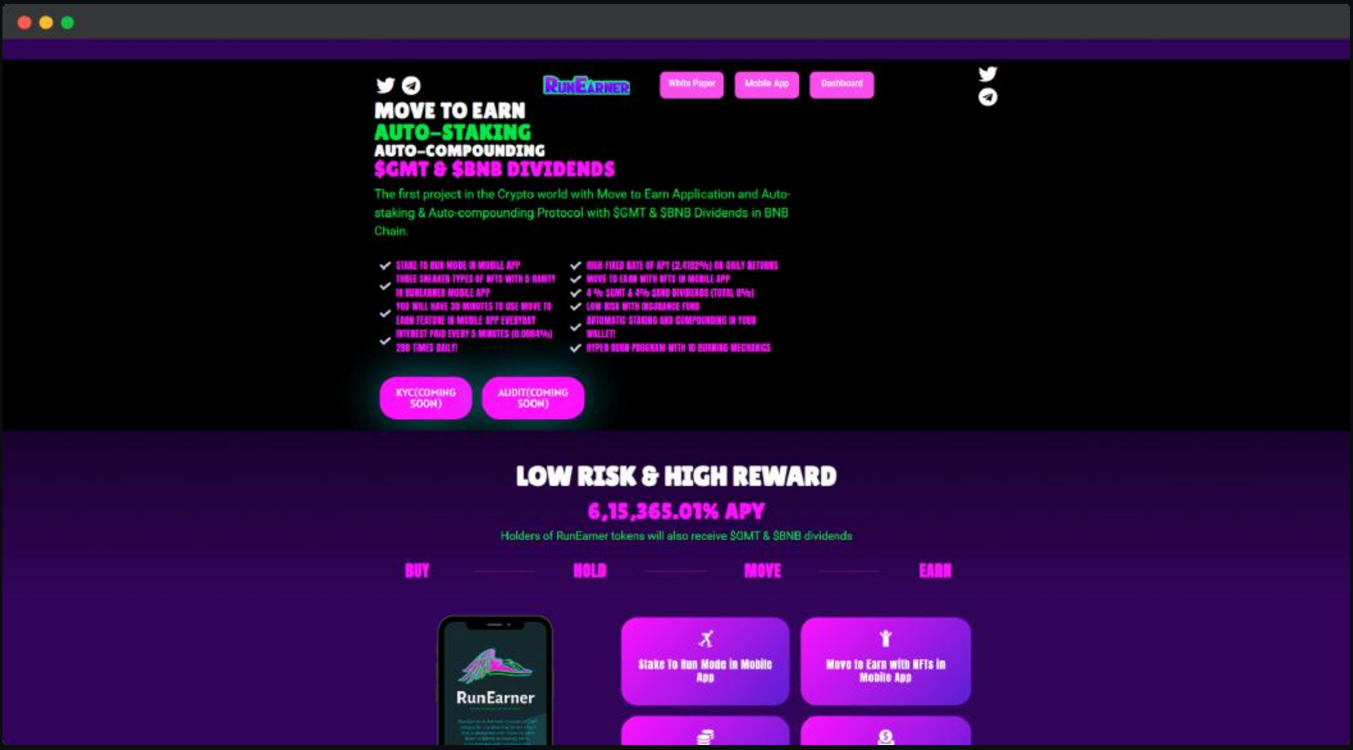
No notes

Contract Snapshot

```
contract RunEarner is ERC20Detailed, Ownable {

using SafeMath for uint256;
using SafeMathInt for int256;

event LogRebase(uint256 indexed epoch, uint256 totalSupply);

IPancakeSwapPair public pairContract;
mapping(address =&gt; bool) _isFeeExempt;

modifier validRecipient(address to) {
    require(to != address(0x0));
    _;
}
```

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.



MOVE TO EARN
AUTO-STAKING
AUTO-COMPOUNDING
$GMT & $BNB DIVIDENDS

The first project in the Crypto world with Move to Earn Application and Auto-staking & Auto-compounding Protocol with $GMT & $BNB Dividends in BNB Chain.

- ✓ STAKE TO RUN MODE IN MOBILE APP
- ✓ THREE SNEAKER TYPES OF NFTS WITH 5 RARITY IN RUNEARNER MOBILE APP
- ✓ YOU WILL HAVE 30 MINUTES TO USE MOVE TO EARN FEATURE IN MOBILE APP EVERYDAY
- ✓ INTEREST PAID EVERY 5 MINUTES (0.0084%) 288 TIMES DAILY!
- ✓ HIGH FIXED RATE OF APY (2.4192%) ON DAILY RETURNS
- ✓ MOVE TO EARN WITH NFTS IN MOBILE APP
- ✓ 4 % $GMT & 4% $BNB DIVIDENDS (TOTAL 8%)
- ✓ LOW RISK WITH INSURANCE FUND
- ✓ AUTOMATIC STAKING AND COMPOUNDING IN YOUR WALLET!
- ✓ HYPER BURN PROGRAM WITH 10 BURNING MECHANICS

LOW RISK & HIGH REWARD

6,15,365.01% APY

Holders of RunEarner tokens will also receive $GMT & $BNB dividends

BUY
HOLD
MOVE
EARN
RunEarner
Stake To Run Mode in Mobile App
Move to Earn with NFTs in Mobile App

● Mobile Friendly

● Does not contain jQuery errors

● SSL Secured

● No major spelling errors

Project Overview

● Not KYC verified by Coinsult

RunEarner

Audited by Coinsult.net

Date: 21 August 2022

✓ Advanced Manual Smart Contract Audit