



Coinsult

Advanced Manual Smart Contract Audit



Project: Griffin Land

Website: <https://griffin.land/>

Low-Risk

4 low-risk code
issues found

Medium-Risk

0 medium-risk code
issues found

High-Risk

0 high-risk code
issues found

Contract Address

0x59234F46AcdB449d5A43eeFd841Bc07a21613Fc1

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	0x27a7504b247dfc8d3d76e82ee3973d33b564b059	1,000,000,000	100.00%

Source Code

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

<https://bscscan.com/address/0x59234F46AcdB449d5A43eeFd841Bc07a21613Fc1#code>

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

4 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 > 0, "Transfer amount must be greater than zero");
    if (from != owner() && to != owner()) {
        require(amount = minimumTokensBeforeSwap;

        if (!inSwapAndLiquify && swapAndLiquifyEnabled && to == uniswapV2Pair) {
            if (overMinimumTokenBalance) {
                contractTokenBalance = minimumTokensBeforeSwap;
                swapTokens(contractTokenBalance);
            }
        }

        bool takeFee = true;

        //if any account belongs to isExcludedFromFee account then remove the fee
```

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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {  
    _taxFee = taxFee;  
}
```

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.

```
function _msgData() internal view virtual returns (bytes memory) {  
    this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2318  
    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.

● **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 includeInReward(address account) external onlyOwner() {
    require(!_isExcluded[account], "Account is already included");
    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 `SSTOREs`, which might lead to an out-of-gas.

Owner privileges

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

Extra notes by the team

No notes

Contract Snapshot

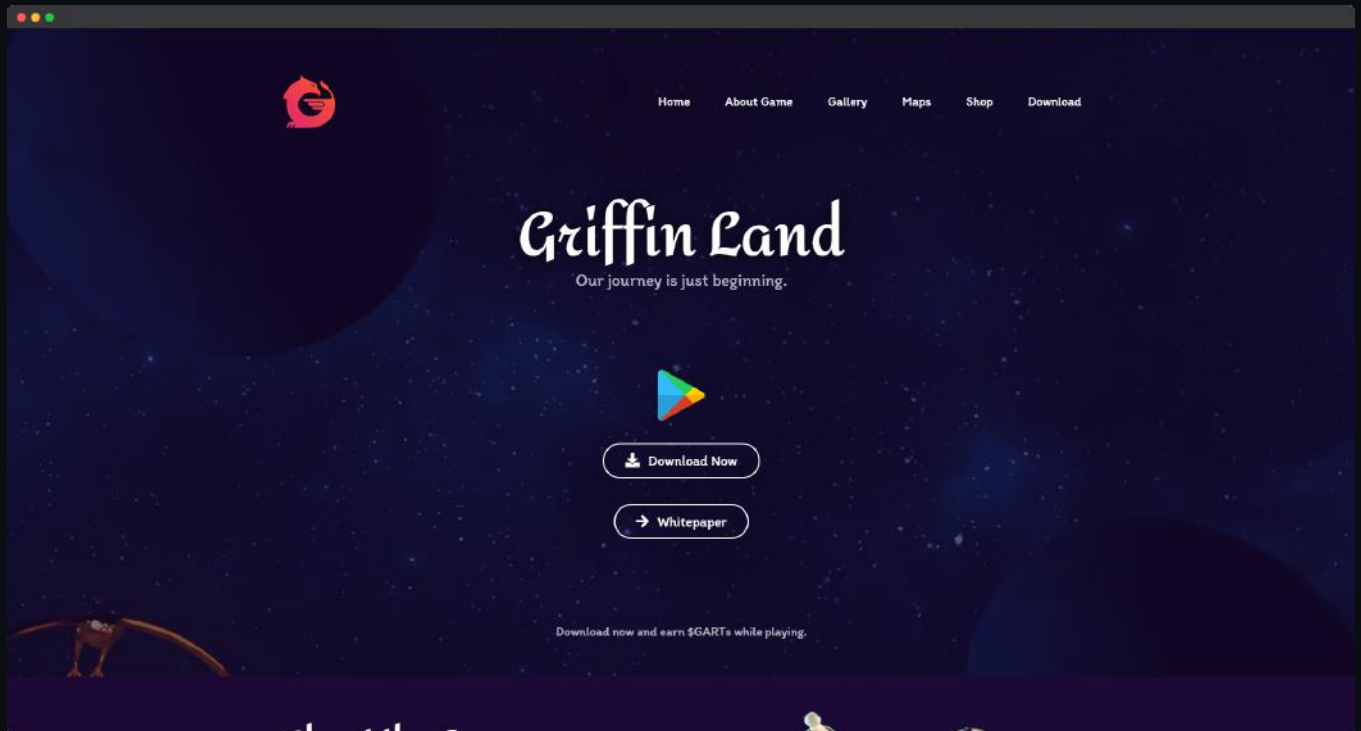
```
contract GriffinLand is Context, IERC20, Ownable {
    using SafeMath for uint256;
    using Address for address;

    address payable public marketingAddress = payable(0x38597d37100ADe36A02F0B28aDe6A9940aCBFe96); // Ma
    address public constant deadAddress = 0x0000000000000000000000000000000000000000000000000000000000000000;
    mapping (address => uint256) private _rOwned;
    mapping (address => uint256) private _tOwned;
    mapping (address => mapping (address => uint256)) private _allowances;

    mapping (address => bool) private _isExcludedFromFee;
```

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

● KYC verified by Coinsult partner

● Not KYC verified by Coinsult

Griffin Land

Completed KYC Verification at a Coinsult partner



✓ Project Owner Identified

✓ Contract: 0x59234F46AcdB449d5A43eeFd841Bc07a21613Fc1

Griffin Land

Audited by Coinsult.net



Date: 27 July 2022

✓ Advanced Manual Smart Contract Audit