Coinsult

Advanced Manual Smart Contract Audit

Project: Shark token

Website: https://www.shark-token.com/

Low-Risk	Medium-Risk	High-Risk
4 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

0xE36232eD368e180D05FFda5E9d05dbaBCb587490

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	0x1acd8ec060bee993611718845330127beb07c54e	500,000,000	100.0000%

Source Code

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

https://bscscan.com/address/0xe36232ed368e180d05ffda5e9d05dbabcb587490#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 &gt; 0, "Transfer amount must be greater than zero");

    // is the token balance of this contract address over the min number of
    // tokens that we need to initiate a swap + liquidity lock?
    // also, don't get caught in a circular liquidity event.
    // also, don't swap &amp; liquify if sender is uniswap pair.
    uint256 contractTokenBalance = balanceOf(address(this));

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

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.

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owner(),
        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.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setTaxFeePercent(uint256 taxFeeBps) external onlyOwner {
    _taxFee = taxFeeBps;
    require(
        _taxFee + _liquidityFee + _charityFee &lt;= 10**4 / 4,
        &quot;Total fee is over 25%&quot;
    );
}
```

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 includeInReward(address account) external onlyOwner {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i &lt; _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 cannot change max transaction amount
- Owner can exclude from fees

Extra notes by the team

No notes

Contract Snapshot

```
contract LiquidityGeneratorToken is IERC20, Ownable, BaseToken {
using SafeMath for uint256;
using Address for address;

uint256 public constant VERSION = 1;

mapping(address =&gt; uint256) private _rOwned;
mapping(address =&gt; uint256) private _tOwned;
mapping(address =&gt; mapping(address =&gt; uint256)) private _allowances;

mapping(address =&gt; bool) private _isExcludedFromFee;
mapping(address =&gt; bool) private _isExcluded;
address[] private _excluded;

uint256 private constant MAX = ~uint256(0);
uint256 private _tTotal;
uint256 private _rTotal;
uint256 private _tFeeTotal;
```

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

Shark token

Audited by Coinsult.net

Date: 24 June 2022

Advanced Manual Smart Contract Audit