HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SMEGMARS
Date: May 25th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for SMEGMARS - Initial Audit
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	BEP20 with reflection and transfer fee
Platform	BSC / Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Deployed mainnet	https://bscscan.com/address/0x62c2a6f57a65e1e4b1d9e31b3e3511c8c36841a8#code
Timeline	21 MAY 2021 - 25 MAY 2021
Changelog	25 MAY 2021 - INITIAL AUDIT

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by SMEGMARS (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on May 25th, 2021.

Scope

The scope of the project is the smart contracts deployed on the Binance Smart Chain mainnet:

`https://bscscan.com/address/0x62c2a6f57a65e1e4b1d9e31b3e3511c8c36841a8#code`

We have scanned these smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Asset’s integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

According to the assessment, the Customer's smart contract is secured but have low issue and some recommendations

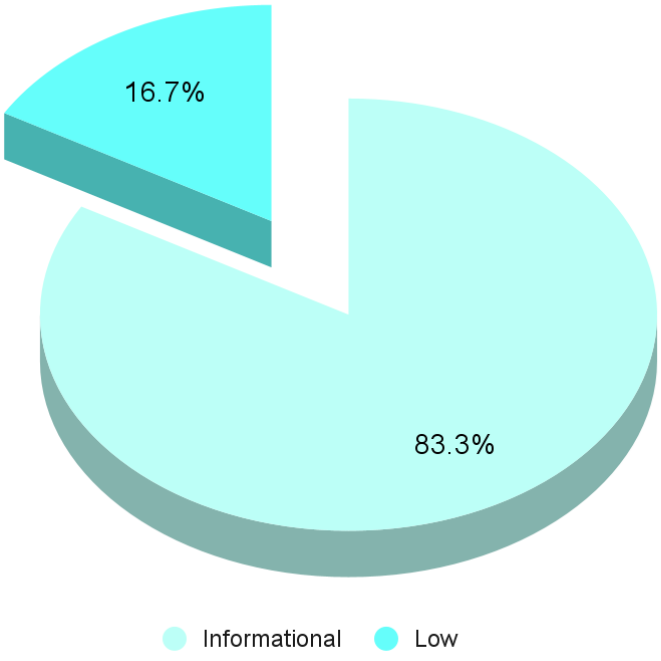
Insecure	Poor secured	Secured	Well-secured

You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

Security engineers found **1** low and **5** informational issues during the first review.

Graph 1. The distribution of vulnerabilities after the first review.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Critical

No Critical severity issues were found.

High

No High severity issues were found.

Medium

No Medium severity issues were found.

Low

1. **Vulnerability:** Excluding non-contract address

   While copying from the ERC to BEP hardcoded address (0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D) of the UniswapV2Router was left in the new contract. In the BSC network this address doesn't represent the Uniswap router, even more, it doesn't have a contract deployed.

   Recommendation: Please consider removing the exclusion or change it to the correct router (pancake, bakery, etc.)

   Lines: #339-340

```
function excludeAccount(address account) external onlyOwner() {
    require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We
can not exclude Uniswap router.');
```

Lowest / Code style / Best Practice

1. **Vulnerability:** Naming convention

   Solidity defines a naming convention that should be followed

   Lines: #499-505

```
function TAXFEE(uint256 taxFee) external onlyOwner() {
    _TAX_FEE = taxFee;
}

function BURNFEE(uint256 burnFee) external onlyOwner() {
    _BURN_FEE = burnFee;
}
```

Lines: #219-221

```
uint256 private _TAX_FEE = 700; // AUTOSTAKING
uint256 private _BURN_FEE = 100; //AUTO BURN
uint256 private _MAX_TX_SIZE = 10000000000 * _DECIMALFACTOR;
```

2. **Vulnerability:** State variable could be declared constant

   state variables that never change its value should be declared **constant** to save gas.

   Lines: #210

```
uint256 private  _DECIMALFACTOR = 10 ** uint256(_DECIMALS);
```

3. **Vulnerability:** View function that could be declared pure

   view functions that never access the contract state should be declared **pure** to save gas.

   Lines: #234-236

```
function name() public view returns (string memory) {
    return _NAME;
}
```

   Lines: #240-242

```
function symbol() public view returns (string memory) {
    return _SYMBOL;
}
```

   Lines: #246-248

```
function decimals() public view returns (uint8) {
    return _DECIMALS;
}
```

4. **Vulnerability:** Too many digits

   Literals with many digits are difficult to read and review.

   Recommendation: Please consider using ether units and/or scientific notation and/or separate with dashes

   ex:
   - 10_000_000_000

- 10e9

Lines: #213

```
uint256 private _tTotal = 10000000000 * _DECIMALFACTOR; // amount of
all tokens
```

Lines: #221

```
uint256 private _MAX_TX_SIZE = 10000000000 * _DECIMALFACTOR;
```

5. **Vulnerability:** Public function that could be declared external

   public functions that are never called by the contract should be declared **external** to save gas.

Lines: #234

```
function name() public view returns (string memory) {
```

Lines: #240

```
function symbol() public view returns (string memory) {
```

Lines: #246

```
function decimals() public view returns (uint8) {
```

Lines: #252

```
function totalSupply() public view override returns (uint256) {
```

Lines: #258

```
function balanceOf(address account) public view override returns
(uint256) {
```

Lines: #265

```
function transfer(address recipient, uint256 amount) public override
returns (bool) {
```

Lines: #272

```
function allowance(address owner, address spender) public view override
returns (uint256) {
```

Lines: #278

```
function approve(address spender, uint256 amount) public override
returns (bool) {
```

Lines: #283

```
function transferFrom(address sender, address recipient, uint256
amount) public override returns (bool) {
```

Lines: #290

```
function increaseAllowance(address spender, uint256 addedValue) public
virtual returns (bool) {
```

Lines: #295

```
function decreaseAllowance(address spender, uint256 subtractedValue)
public virtual returns (bool) {
```

Lines: #300

```
function isExcluded(address account) public view returns (bool) {
```

Lines: #304

```
function totalFees() public view returns (uint256) {
```

Lines: #308

```
function totalBurn() public view returns (uint256) {
```

Lines: #312

```
function deliver(uint256 tAmount) public {
```

Lines: #321

```
function reflectionFromToken(uint256 tAmount, bool deductTransferFee)
public view returns(uint256) {
```

Lines: #510

```
function burn(uint256 amount) public {
```

3. Lines 283, 285, 295, 296, 321, 340, 393, 403, 414, 425, 442, 443, 445, 459, 466 and 483 are above the recommended maximum line length.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **1** low and **5** informational issues during the first review.

Category	Check Items	Comments
➔ Code Review	➔ Style guide violation	➔ maximum line length ➔ too many digits
	➔ Gas savings	➔ public function that could be declared external ➔ view function that could be declared pure ➔ state variable that could be declared constant
➔ Functional review	➔ Business Logics Review	➔ Hard-coded non-existent contract

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.