HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SafeEarth
Date: April 11th, 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 SafeEarth - Initial Audit
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 Token / ETH Swap
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Deployed code	https://etherscan.io/address/0xe6f1966d04cfcb9cd1b1dc4e8256d8b501b11cba#code
Timeline	8 APRIL 2021 - 11 APRIL 2021
Changelog	11 APRIL 2021 - INITIAL AUDIT

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

Table of contents

Introduction 4
Scope 4
Executive Summary 5
Severity Definitions 6
AS-IS overview 7
Conclusion 9
Disclaimers 10

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

Introduction

Hacken OÜ (Consultant) was contracted by SafeEarth (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 April 11th, 2021.

Scope

The scope of the project is a smart contract deployed in the Ethereum network:

https://etherscan.io/address/0xe6f1966d04cfcb9cd1b1dc4e8256d8b501b11cba#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.

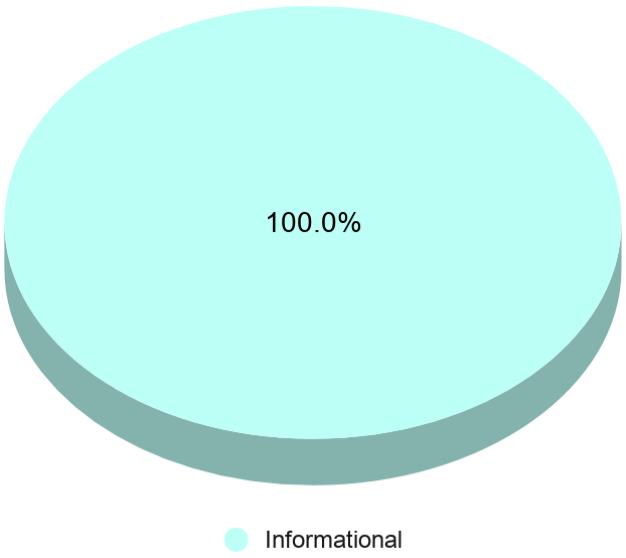
Insecure	Poor secured	Secured	Well-secured

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. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found **4** 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

No Low severity issues were found.

Lowest / Code style / Best Practice

1. **Vulnerability:** Too many digits
   Contracts: SAFEEARTH

   Literals with many digits are difficult to read and review. Please consider using *scientific notation* and *ether units* for better readability. Ex. instead of 1000000000 * 10**6 * 10**9 try the following:
   - 1e6 ether
   - 1e9 finney
   - 1e15 * 10**_digits
   - or even 1_000_000_000 * 1e6 * 10**_digits

   Lines: SAFEEARTH.sol#225

```
uint256 private _tTotal = 1000000000 * 10**6 * 10**9;
```

   Lines: SAFEEARTH.sol#247-248

```
uint256 public _maxTxAmount = 5000000 * 10**6 * 10**9;
uint256 private numTokensSellToAddToLiquidity = 500000 * 10**6 * 10**9;
```

2. **Vulnerability:** State variables that could be declared constant
   Contract: SAFEEARTH

   State variables that never change their values should be declared constant to save gas.

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

Lines: SAFEEARTH.sol#225

```
uint256 private _tTotal = 1000000000 * 10**6 * 10**9;
```

Lines: SAFEEARTH.sol#229-231

```
string private _name = "SafeEarth";
string private _symbol = "SAFEEARTH";
uint8 private _decimals = 9;
```

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

Contracts: SAFEEARTH

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

Lines: SAFEEARTH.sol#279

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

Lines: SAFEEARTH.sol#283

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

Lines: SAFEEARTH.sol#287

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

Lines: SAFEEARTH.sol#291

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

Lines: SAFEEARTH.sol#300

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

Lines: SAFEEARTH.sol#305

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

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

Lines: SAFEEARTH.sol#309

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

Lines: SAFEEARTH.sol#314

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

Lines: SAFEEARTH.sol#320

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

Lines: SAFEEARTH.sol#325

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

Lines: SAFEEARTH.sol#330

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

Lines: SAFEEARTH.sol#334

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

Lines: SAFEEARTH.sol#338

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

Lines: SAFEEARTH.sol#347

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

Lines: SAFEEARTH.sol#364

```
function excludeFromReward(address account) public onlyOwner() {
```

Lines: SAFEEARTH.sol#396

```
function excludeFromFee(address account) public onlyOwner {
```

Lines: SAFEEARTH.sol#400

```
function includeInFee(address account) public onlyOwner {
```

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

Lines: SAFEEARTH.sol#426

```
function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
```

Lines: SAFEEARTH.sol#515

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

4. Lines 267, 269, 314, 316, 325, 326, 347, 386, 408, 442, 444, 455, 472, 551, 634, 643, 653 of the SAFEEARTH.sol 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. For the contract, high-level description of functionality was presented in As-Is overview section of the report.

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

Security engineers found **4** informational issues during the first review.

Category	Check Items	Comments
➔ Code Review	➔ Style guide violation	➔ Public function that could be declared external ➔ Maximum Line Length ➔ State variables that could be declared constant ➔ Too many digits

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 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.