HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Foil Network
Date: September 3rd, 2021

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

Document

Name	Smart Contract Code Review and Security Analysis Report for Foil Network.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Seed contract
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://gitlab.com/FoilNetwork/tge-seed-round-smart-contract
Commit	D3522A4628CE735058E57852AB1425600D296FC8
Technical Documentation	NO
JS tests	YES
Timeline	19 JULY 2021 - 03 SEPTEMBER 2021
Changelog	21 JULY 2021 - INITIAL AUDIT 05 AUGUST 2021 - SECOND REVIEW 17 AUGUST 2021 - THIRD REVIEW 03 SEPTEMBER 2021 - FOURTH REVIEW

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 Foil Network (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between July 19th, 2021 - July 21th, 2021. The second code review conducted on August 5th, 2021. The third code review conducted on August 17th, 2021. The fourth code review conducted on September 3rd, 2021.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://gitlab.com/FoilNetwork/tge-seed-round-smart-contract
Commit:
d3522a4628ce735058e57852ab1425600d296fc8
Technical Documentation: No
JS tests: Yes
Contracts:
contracts/FoilSeedContract.sol
contracts/Lockup.sol
test/sample-test.js

We have scanned this smart contract 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 ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are well-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. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found 2 critical, 1 high, 4 medium, 10 low severity issues during the first review.

Security engineers found 2 medium, 3 low severity issues during the second review.

Security engineers found 2 medium issues during the third review.

After the latest fourth review security engineers found that all issues were fixed and the current code contains **no more issues**.

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

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

Audit overview

Critical

1. Non-revertible change of state variable

   Once someone buys with **USDT**, buyType will become '*USDT*' and will never be changed back to '*ETH*', so every next call to **lockUp.deposit** will send '*USDT*' as the second argument, as well as **lockUp.withdraw**. The same issue happens when someone claims from the **USDT** side.

   Fixed before the second review.

2. No token transfers

   A function **buyFoilToken**, which should definitely deal with tokens, doesn't do any transfers. It just does calculations and then calls **buyCommon** which has commented-out calls to some transfer functions.

   Recommendation: Please re-check the entire business logic in the code.

   Fixed before the second review.

High

One of two tests failed

Running provided tests, which we found only two, one of them failing.

Recommendation: Please fix the tests. Also, try to cover as much as possible contract code.

Fixed before the second review.

Medium

1. Ignored result of SafeMath.sub function

   SafeMath.sub function returns the result of the subtraction operation therefore this result should be taken to deal with it.

   Recommendation: Please consider using the return value of the function or remove this call.
   Second review: After switching from using the SafeMath library this point became weirder. **foilToken.balanceOf** result is being subtracted with **pendingToken** in the **unchecked** block (that could result in underflow) but the result is never used after that.

Third review: Still no reason to have those lines (136) which aren't doing anything except burning transaction gas.

Recommendation: Please remove those lines completely.

Fixed before the fourth review.

2. Ignored result of SafeMath.add function

SafeMath.add function returns the result of the addition operation therefore this result should be taken to deal with it.

Recommendation: Please consider using the return value of the function or remove this call.

Second review: After switching from using the SafeMath library this point became weirder. **foilToken.balanceOf** result is being added by **pendingToken** in the **unchecked** block (that could result in underflow) but the result is never used after that.

Third review: Still no reason to have those lines (137) which aren't doing anything except burning transaction gas.

Recommendation: Please remove those lines completely.

Fixed before the fourth review.

3. Ignored result of IERC20.transfer function

IERC20.transfer function returns a boolean result of the token transfer operation. To be sure that tokens were transferred successfully, you should check the return value.

Recommendation: Please consider checking the return value of the function.

Fixed before the second review.

4. Contracts that lock Ether

Contract with a **payable** function, but without a withdrawal capacity. Every Ether sent to **FoilSeedContract** will be lost.

Recommendation: Remove the payable attribute or add a withdraw function.

Fixed before the second review.

Low

1. Missing zero address validation

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

The constructor lacks zero-check for _foilWallet address.

Fixed before the second review.

2. Block timestamp

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

Recommendation: Avoid relying on block.timestamp. You may try to rely on the block.number instead.

Second review: Just changing block.timestamp to block.number in the code doesn't do the trick. A new block is not generated each second, so you couldn't rely on that 1 minute will equal 60 blocks!

Fixed before the third review.

3. Multiple access to same state variable

There are two calls to the function **isWhitelisted** in the same function. it's cheaper to store the result in the local variable.

Fixed before the second review.

4. Excess check

A modifier **initialized** will always pass because there is no way for the foilWallet to be a zero-address as it assigned in the constructor with pre-check.

Fixed before the second review.

5. Missing events

All critical changes should emit events so it could be easily tracked.

Recommendation: Please consider adding events for all: changing maxBuyLimit, finalizing, whitelisting, claiming, buying, etc.

Fixed before the second review.

6. Unused state variable

The FoilSeedContract contract declares a private state variable **cliamedTokens** which is never used in the code.

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

Recommendation: Please consider removing the variable declaration

Fixed before the second review.

7. SafeMath in solidity >0.8

   Starting solidity version 0.8 and later math is already safe from the over and underflow. And it's not needed to check the result anymore.

   Recommendation: Please consider not using SafeMath with solidity >= 0.8

   Second Review: Replacing SafeMath calls by unchecked blocks removes any over/underflow checks, built in the solidity starting 0.8.0

   Recommendation: Please remove unchecked blocks and leave pure math.

   Fixed before the third review.

8. State variables that could be declared constant

   Constant state variables should be declared constant to save gas.

   Fixed before the second review.

9. Public function that could be declared external

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

   Fixed before the second review.

10. Maximum line length

    Lines: 173, 183, 188, 193, 198, 203, 208, 213, 279, 302 and 305 are exceeding recommended solidity maximum line length

    Fixed before the third review.

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.

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

As a result of the audit, security engineers found 2 critical, 1 high, 4 medium, 10 low severity issues during the first review.

Security engineers found 2 medium, 3 low severity issues during the second review.

Security engineers found 2 medium issues during the third review.

After the latest fourth review security engineers found that all issues were fixed and the current code contains **no more issues**.

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 the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free 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 the security of smart contracts.

Technical Disclaimer

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