SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SafeGram
Date: March 24th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for SafeGram.
Approved by	Andrew Matiukhin | CTO Hacken OU Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Custom BEP-20 token
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/Safegramhub/Farms
Commit	6c99ba17fdddb5f4a8335a3ef555e9ba22aa98c4
Technical Documentation	YES
JS tests	YES
Website	
Timeline	8 MARCH 2022 - 24 MARCH 2022
Changelog	10 MARCH 2022 - Initial Review 24 MARCH 2022 - Revising

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by SafeGram (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 contracts.

The second review was conducted on March 24th, 2022.

Scope

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

Repository:
    https://github.com/Safegramhub/Farms
Commit:
    6c99ba17fdddb5f4a8335a3ef555e9ba22aa98c4
Technical Documentation: Yes
JS tests: Yes (low coverage)
Contracts:
    ./contracts/GramToken.sol

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 ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer did not provide functional or technical requirements, only the plan of token distribution is provided. The total Documentation Quality score is **3** out of **10**. The weight in the total score is **1**.

After the second audit, the total documentation quality score is **10** out of **10**. The Customer provided requirements documentation.

Code quality

The code follows official language style guides. Code has no comments. Code is not covered with unit tests. The score is **3** out of **10**. The weight in the total score is **1**.

After the second audit, the code quality score is **5** out of **10**. The code was significantly reworked. The code has low unit test coverage.

Architecture quality

Smart contract of the project has unclear architecture and redundant code. The architecture quality score is **1** out of **10**. The weight in the total score is **1**.

After the second audit, the architecture quality score is **10** out of **10**. The code was reworked, redundant logic was removed. The project has clear and clean architecture.

Security score

As a result of the audit, security engineers found **3** low, **1** medium and **2** high severity issues. The security score is **0** out of **10**. All found issues are displayed in the “Issues overview” section. The weight in the total score is **7**.

After the second audit, **no** issues were found. The security score is **10** out of **10**.

Summary

According to the assessment, the Customer's smart has the following score: **9.5**

1
2
3
4
5
6
7
8
9
10
You are here

Notices

1. Staking logic is out of audit scope, only token code is audited.

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 cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Audit overview

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

1. The contract logic does not update the value of the `_isStakingShared` flag. It makes the `_masterChef` address possible to send almost all the tokens to staking.

   Contracts: GramToken.sol

   Function: getStakingShare

   Recommendation: Set `_isStakingShared` to true after the tokens are sent once

   Status: Fixed

2. The contract has a `reflect` function, which burns reflected tokens of the sender in proportion to non-reflected tokens. The function does not check if the sender has enough non-reflected tokens;

   Contracts: GramToken.sol

   Function: reflect

   Recommendation: Add a conditional statement to check if the sender has sufficient non-reflected tokens.

   Status: Fixed

🟩🟩 Medium

1. The contract fee is set to 0, which means that the contract has redundant logic.

   The functions `_getTValues` and `_getPTValues` are redundant. The functions set `tFee` to 0. Due to this logic the functions always return `tAmount` equal to `tTransferAmount` and `tFee` equal to 0.

   The functions `_getRValues` and `_getPValues` have redundant return fields. The functions are bound to `_getPTValues` and `_getTValues` logic, the functions always return `rFee` equal to 0 and `rAmount` returns value equal to `rTransferAmount`.

   The function `_getValues` has redundant return fields. The function is bound to `_getTValues` and `_getRValues` logic, it always returns `rFee` and `tFee` equals to 0 and `rAmount` value equals to `rTransferAmount`.

   The function `_reflectFee` is redundant, because the contract has no transaction fee inside the `_getTValues` function.

The function `reflectionFromToken` contains unnecessary condition statement in case of 0 fee.

The contract field `_tFeeTotal` is not used in the contract.

Contracts: GramToken.sol

Function: _getTValues, _getPTValues, _getPValues, _getRValues, _getValues, _reflectFee, reflectionFromToken

Recommendation: Remove the logic related to fees.

Status: Fixed

Low

1. The contract function has a redundant condition statement.

   Contracts: GramToken.sol

   Function: _transfer

   Recommendation: Remove redundant if condition statement `!_isExcluded[sender] && !_isExcluded[recipient]`.

   Status: Fixed

2. The contract function has a typo in the error message for the required statement.

   Contracts: GramToken.sol

   Function: includeAccount

   Recommendation: Replace “Account is already excluded” with “Account is already included”

   Status: Fixed

3. The contract has public functions which are never used internally.

   Contracts: GramToken.sol

   Function: getStakingShare

   Recommendation: Replace function visibility with external

   Status: Fixed

Recommendations

1. It is recommended to use a recent version of the Solidity compiler.

   Contracts: GramToken.sol

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with 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 a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. 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 cannot guarantee the explicit security of the audited smart contracts.