RD AUDITORS

SOLSTER SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer:	Solster
Prepared on:	23 September 2021
Platform:	Solana
Language:	RUST

.

TABLE OF CONTENTS

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND 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 THE DECISION OF CUSTOMER.

Document

Name	**Smart Contract Code Review and Security Analysis Report of Solster**
Platform	Solana / RUST
File/Folder	ido_program.rs
MD5 hash	8C153C9C9207844BBCBAA11535ECB6B4
SHA256 hash	AA69056ADC0F97A4B51551F3866264D8B645DF5B3EA950EF56A98078418476BC
File/Folder	stake.rs
MD5 hash	5F97C173DA8C71DD688F0A404FE23A8B
SHA256 hash	0EBB9FB0A2C3A7CE4DCEF65019276D829EC50B9AA76B6847FD96B427A5174C93
Date	23/09/2021

Introduction

RD Auditors (Consultant) were contracted by Solster (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between
17 - 23 September 2021.

This contract consists of one project folder with components and supporting files in the form of solster.zip.

Project Scope

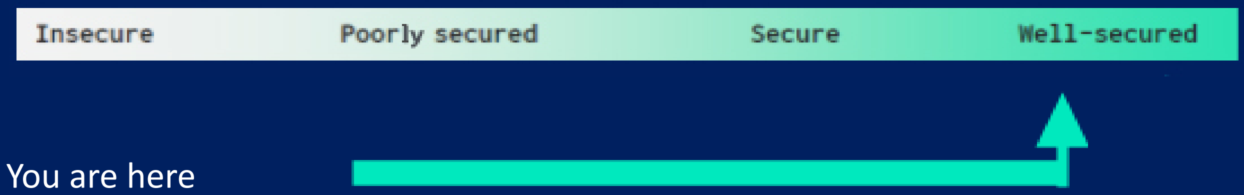
The scope of the project is a smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to) under the scope of the SOLANA architecture.

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer’s RUST smart contract is well secured.



Manual and localised checks are done. All issues were performed by our team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

Please find a link that, within this report, uses RUST libraries specially designed for smart contracts taken from the popular open source.

The libraries/modules within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times with parallel execution offered by SOLANA blockchain or by other contracts.

The IDO could have some scenario and unit test code blocks, which would be more helpful to determine the integrity of the code, but providing functional description along with source helped a lot to frame more test cases.

Overall, the code is well commented. Commenting can provide rich documentation for functions, return variables and more.

Documentation

We were given the IDO contract as a zip file.

The hash of that file is mentioned in the table. As mentioned above, It's well commented smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries/modules are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

Solster

IDO smart contract with staking feature.

Entire projects (modules/structure/implementation/methods) are written with proper comments, and in our test and manual checks found it perfect for production. In our assessment this smart contract is well-secured and ready for production.

File And Function Level Report

File: ido_program.rs

Contract: IDO
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Observation	Test Report	Conclusion	Score
1	initialize	No Issue	All Passed	No Issue	Passed
2	depositer	No Issue	All Passed	No Issue	Passed
3	Update_registrar	No Issue	All Passed	No Issue	Passed
4	create_member	No Issue	All Passed	No Issue	Passed
5	Update_member	No Issue	All Passed	No Issue	Passed
6	deposit	No Issue	All Passed	No Issue	Passed
7	stake	No Issue	All Passed	No Issue	Passed
8	start_Unstake	No Issue	All Passed	No Issue	Passed
9	end_Unstake	No Issue	All Passed	No Issue	Passed
10	withdraw	No Issue	All Passed	No Issue	Passed

File: stake.rs

Contract:	Stake
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

Sl.	Function	Observation	Test Report	Conclusion	Score
1	initialize	No Issue	All Passed	No Issue	Passed
2	depositer	No Issue	All Passed	No Issue	Passed
3	Update_registrar	No Issue	All Passed	No Issue	Passed
4	Create_member	No Issue	All Passed	No Issue	Passed
5	Update_member	No Issue	All Passed	No Issue	Passed
6	deposit	No Issue	All Passed	No Issue	Passed
7	stake	No Issue	All Passed	No Issue	Passed
8	start_unstake	No Issue	All Passed	No Issue	Passed
9	end_unstake	No Issue	All Passed	No Issue	Passed
10	withdraw	No Issue	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
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 lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest **Code Style/** **Best Practice**	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

No low severity vulnerabilities were found.

Very Low

No very low severity vulnerabilities were found.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, so it is ready to go for production.

Since possible test cases can be unlimited, and it was good that developer level documentation (function level description) was provided which helped a lot to frame more test cases, for such an extensive smart contract protocol, however we covered more and more test cases but under unlimited possibilities we provide no such guarantee of future outcomes. We have used all the latest known facts/outcomes of SOLANA and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is “well secured”

Note For Contract user

There might be other contracts in their platform which are unaudited.. Owner has full control over the smart contract. Thus, technical auditing does not guarantee the project's ethical side..

Please do your due diligence before investing. Our audit report is never an investment advice.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:
In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:
Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed 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).

Because the total number of test cases are unlimited, 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, 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. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

RD
AUDITORS

Email: info@rdauditors.com

Website: www.rdauditors.com