BlockApex

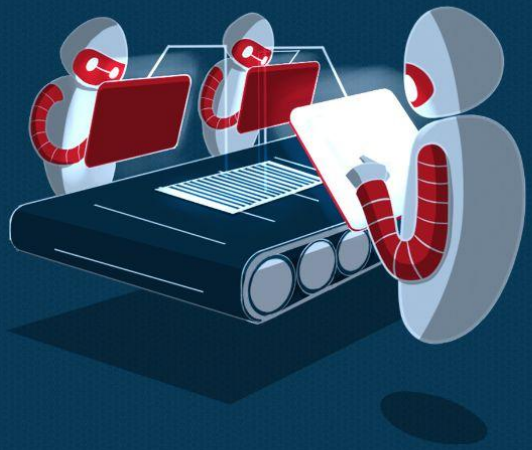
SMART CONTRACT SECURITY ANALYSIS REPORT

```
pragma solidity 0.7.0;
contract Contract {

    function hello() public returns (string) {
        return "Hello World!";
    }

    function findVulnerability() public returns (string) {
        return "Finding Vulnerability";
    }

    function solveVulnerability() public returns (string) {
        return "Solve Vulnerability";
    }
}
```



Powered by XORD

PREFACE

Objectives

The purpose of this document is to highlight any identified bugs/issues in the provided codebase. This audit has been conducted in a closed and secure environment, free from influence or bias of any sort. This document may contain confidential information about IT systems/architecture and intellectual property of the client. It also contains information about potential risks and the processes involved in mitigating/exploiting the risks mentioned below. The usage of information provided in this report is limited, internally, to the client. However, this report can be disclosed publicly with the intention to aid our growing blockchain community; under the discretion of the client.

Key understandings

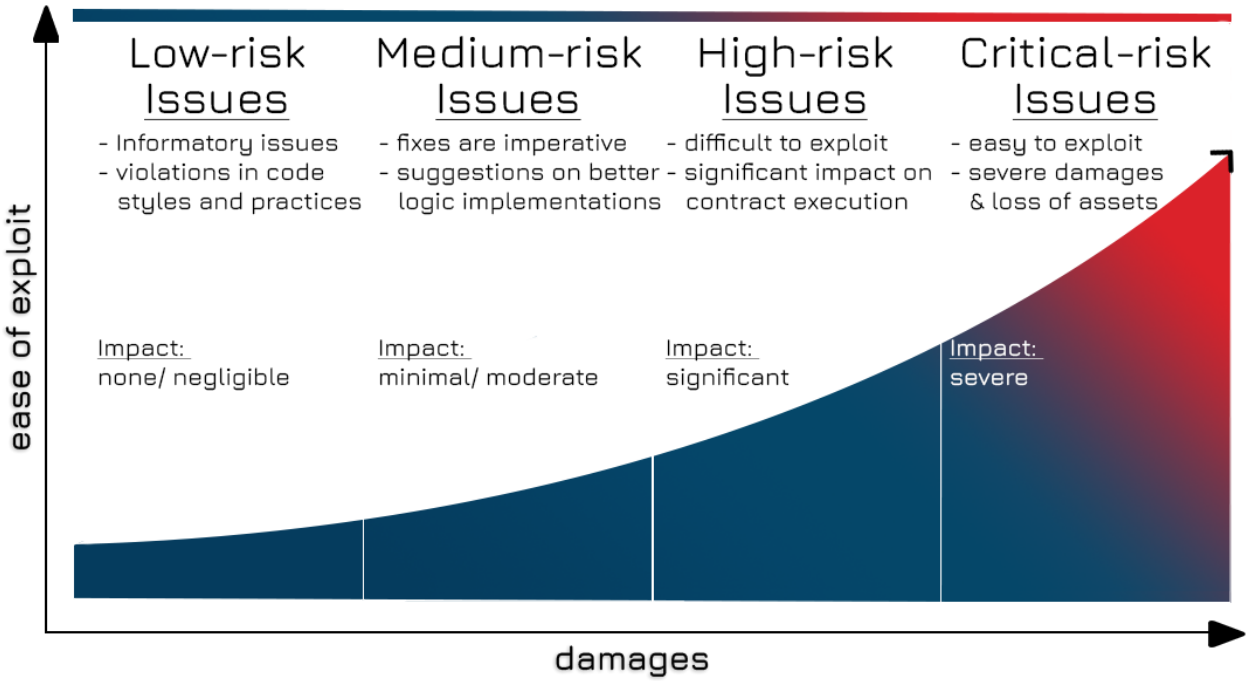


TABLE OF CONTENTS

INTRODUCTION

BlockApex (Auditor) was contracted by BullionFX (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which started on 20th Nov 2022.

Name
DEMI (Rain Protocol)
Auditor
BlockApex
Platform
EVM-based / Solidity
Type of review
Manual Code Review | Automated Tools Analysis
Methods
Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git repository/ Commit Hash
e2277e2253c60b04924f52b68e4ab6df4a68df6e
White paper/ Documentation
Docs
Document log
Initial Audit Completed: Nov. 27th, 2022
Final Audit: Nov 29th, 2022

Scope

The git-repository shared was checked for common code violations along with vulnerability-specific probing to detect **major issues/vulnerabilities**. Some specific checks are as follows:

Code review		Functional review
Reentrancy	Unchecked external call	Business Logics Review
Ownership Takeover	FT token API violation	Functionality Checks
Timestamp Dependence	Unchecked math	Access Control & Authorization
Gas Limit and Loops	Unsafe type inference	Escrow manipulation
DoS with (Unexpected) Throw	Implicit visibility level	Token Supply manipulation
DoS with Block Gas Limit	Deployment Consistency	Asset's integrity
Transaction-Ordering Dependence	Repository Consistency	User Balances manipulation
Style guide violation	Data Consistency	Kill-Switch Mechanism
Costly Loop	Inheritance Ordering	Operation Trails & Event Generation

Project Overview

Rain Protocol is a set of building blocks that enable your token economy including staking, vesting, emissions, escrow, order book, verification, sale and membership. Rain VM uses EVM making it fully compatible with all the EVM chains. DEMI has integrated Rain protocol's Staking contracts.

System Architecture

Staking Factory Contract

The system uses a design pattern called 'factory method' to deploy homogenous contracts from the same parent & keeps record of the child contracts that have been deployed by the factory. In this particular case, the system deploys a staking contract.

Staking Contract

Staking contract is a modified version of a vault contract tokenized. It inherits all of the properties of ERC-4626 contract which standardizes the interface for easily managing deposited tokens & their shares within the system. The contract also introduces a custom logic for maintaining it's own internal ledger for share calculation through the checkpointing mechanism.

Methodology & Scope

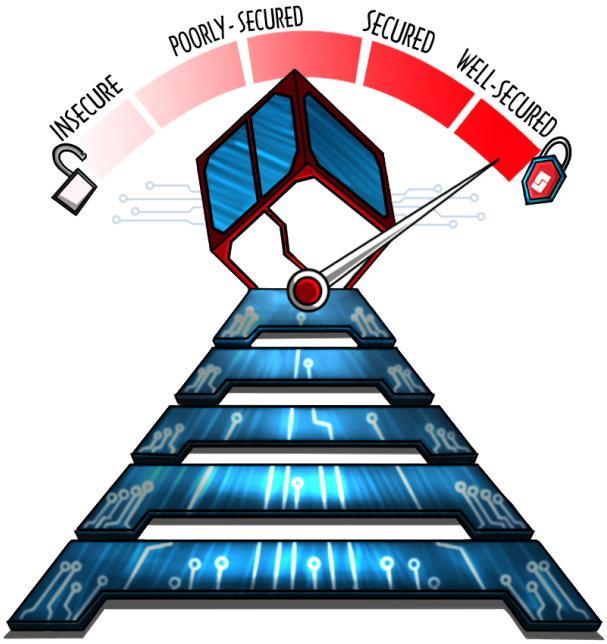
The codebase was audited using a filtered audit technique. A pair of two (2) auditors scanned the codebase in an iterative process spanning over a time of One (1) weeks

Starting with the recon phase, a basic understanding was developed and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews with the motive to find logical flaws in the codebase complemented with code optimizations, software, and security design patterns, code styles, best practices, and identifying false positives that were detected by automated analysis tools.

AUDIT REPORT

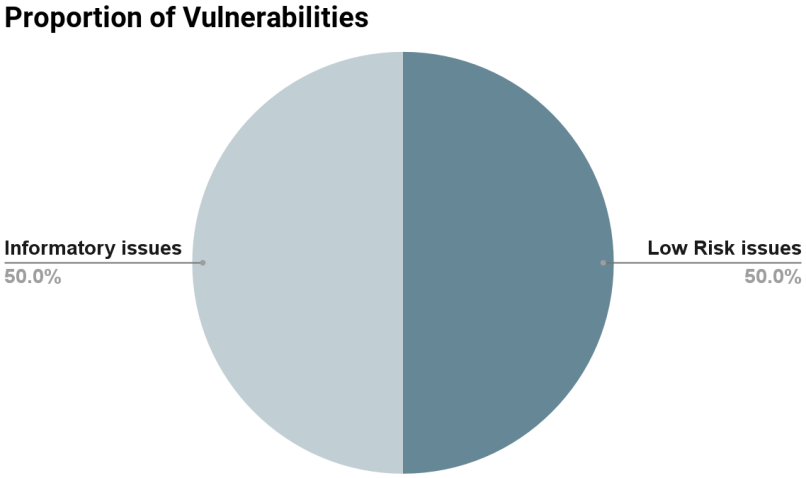
Executive Summary

Our team performed a technique called “Filtered Audit”, where the contract was separately audited by four individuals. After a thorough and rigorous process of manual testing, an automated review was carried out using slither for static analysis. All the flags raised were manually reviewed and re-tested to identify the false positives.



Our team found:

# of issues	Severity of the risk
0	Critical Risk issue(s)
0	High Risk issue(s)
0	Medium Risk issue(s)
1	Low Risk issue(s)
1	Informatory issue(s)



Key Findings

#	Findings	Risk	Status
1	Potential Loss of Precision	Low	Acknowledged
2	Inexplicable inclusion of unsed library	Informational	Fixed

Detailed Overview

Critical-risk issues

No critical issues found.

High-risk issues

No High-risk issues found.

Medium-risk issues

No Medium-risk issues found.

Low-risk issues

1. Potential Loss of Precision

File: stake/Stake.sol#L15

Description:

In `contracts/math/FixedPointMath.sol` the function `scaleN` will lead to precision loss if a number is scaled down to 18.
There may arise a scenario in future development where a deposited token having higher decimal (i.e > 18) may need to scale down to 18 decimals for logic consistency. The conversion either from or to 18 will lead to loss in precision which may result in dust amount being locked in the contract.
However, this function is not used in the current implementation of the staking contract to either scale up or down.

Recommendation: Introduce a mechanism to manage the difference that was lost during the conversion. One way could be by storing the difference & using it to convert back.

Dev's Response

As noted staking contract does not use scaleN. Worth also noting that ERC4626 that the staking contract is based on dedicates a lot of the spec to rounding/dust handling, so if scaleN would be hypothetically used in the future it would still need to be 4626 compliant (which means always leaving dust from the underlying asset in the vault in the case of rounding issues) that's largely what the mulDiv is handling in openzeppelin's implementation and we wrap in the other fixed point math functions

As scaleN is a library contract it has no storage of its own so there's nowhere for it to save information about the lost precision directly, the best it could do is return two values, one representing the scaled value and one representing the lost precisioN. Currently scaleN is only used in expressions in the interpreter as a provided opcode, so i'm not sure if it's something on the average expression author's radar to be worrying about or ready to handle (juggling 2 values on the stack to avoid some dust).

it's probably worth documenting all this though as it's worth pointing out as something to consider for anyone who does care about it.

Auditor's Response
The auditor's agree with the devs.

Informatory issues and Optimizations

1. Inexplicable inclusion of unsed library

File: stake/Stake.sol#L15

Description:

The staking contract imports a library called `FixedPointMath.sol` that is not used within the scope of the contract.

Recommendation: Remove the unused import along with its using statement.

```
// import "../math/FixedPointMath.sol";
// using FixedPointMath for uint256;
```

Alleviation: This issue is **fixed**.

DISCLAIMER

The smart contracts provided by the client for audit purposes have been thoroughly analyzed in compliance with the global best practices till date w.r.t cybersecurity vulnerabilities and issues in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token, its sale or any other aspect of the project.

Crypto assets/tokens are results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third-party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks.

This audit 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 security of smart contracts.