BLOCK AUDIT REPORT

Smart Contract Security Audit Report

JERRY

Block Audit Report Team received the **Jerry** team's application for smart contract security audit of the **Jerry** Token on June 03, 2021. The following are the details and results of this smart contract security audit:

Token Name: **Jerry**

The Contract address: 0x8fdB928DbBa0B1833d20e24cE934EA4594B3Adb3

Link Address:
https://bscscan.com/address/0x8fdB928DbBa0B1833d20e24cE934EA4594B3Adb3#code

The audit items and results:

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

Audit Number: BAR0030062021

Audit Date: June 04, 2021

Audit Team: Block Audit Report Team

Table of Content

Introduction **4**
Auditing Approach and Methodologies applied 4
Audit Details 4

Audit Goals **5**
Security 5
Sound Architecture 5
Code Correctness and Quality 5

Security **5**
High level severity issues 5
Medium level severity issues 5
Low level severity issues 5
Issues checking status 6

Manual Audit: **7**
Critical level severity issues 7
High level severity issues 7
Medium level severity issues 7
Low level severity issues 7

Automated Audit **8**
Remix Compiler Warnings 8

Disclaimer **9**

Summary **9**

Introduction

This Audit Report mainly focuses on the overall security of **Jerry** Smart Contract. With this report, we have tried to ensure the reliability and correctness of their smart contract by complete and rigorous assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Block Audit Report team has performed rigorous testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted custom unit tests written for each function in the contract to verify that each function works as expected.

In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analyzing the complexity of the code in depth and detailed, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

Audit Details

Project Name: **Jeery**
Website/ Bscscan Code **(Mainnet):**
0x8fdB928DbBa0B1833d20e24cE934EA4594B3Adb3
Languages: Solidity (Smart contract)

Platforms and Tools: Achilles, Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Securify, Mythril, Contract Library

Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient and working according to the specifications. The audit activities can be grouped in the following three categories:

Security

Identifying security related issues within each contract and the system of contract.

Sound Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Issue Categories

Every issue in this report was assigned a severity level from the following:

High level severity issues

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low level severity issues

Issues on this level are minor details and warningJs that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

Critical	High	Medium	Low	Note
0	0	1	1	0

Issues Checking Status

№	Issue description.	Checking status
1	Compiler warnings.	Medium Issues
2	Race conditions and Reentrancy. Cross-function race conditions.	Passed
3	Possible delays in data delivery.	Passed
4	Oracle calls.	Passed
5	Front running.	Passed
6	Timestamp dependence.	Passed
7	Integer Overflow and Underflow.	Passed
8	DoS with Revert.	Passed
9	DoS with block gas limit.	Passed
10	Methods execution permissions.	Passed
11	Economy model.	Passed
12	The impact of the exchange rate on the logic.	Passed
13	Private user data leaks.	Passed
14	Malicious Event log.	Passed
15	Scoping and Declarations.	Passed
16	Uninitialized storage pointers.	Passed
17	Arithmetic accuracy.	Passed
18	Design Logic.	Passed
19	Cross-function race conditions.	Passed
20	Safe Zeppelin module.	Passed
21	Fallback function security.	Passed

Manual Audit:

For this section the code was tested/read line by line by our developers. We also used Remix IDE's JavaScript VM and Kovan networks to test the contract functionality.

Critical Severity Issues

No critical severity issues found.

High Severity Issues

No high severity issues found.

Medium Severity Issues

1. SWC-102 / lines: 9

```
Medium
-       A security vulnerability has been detected.
8
9           pragma solidity ^0.4.24;
10
```

In detail:

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Low Severity Issues

1. SWC-103 / lines: 9

```
Low
-       A security vulnerability has been detected.
8
9           pragma solidity ^0.4.24;
10
```

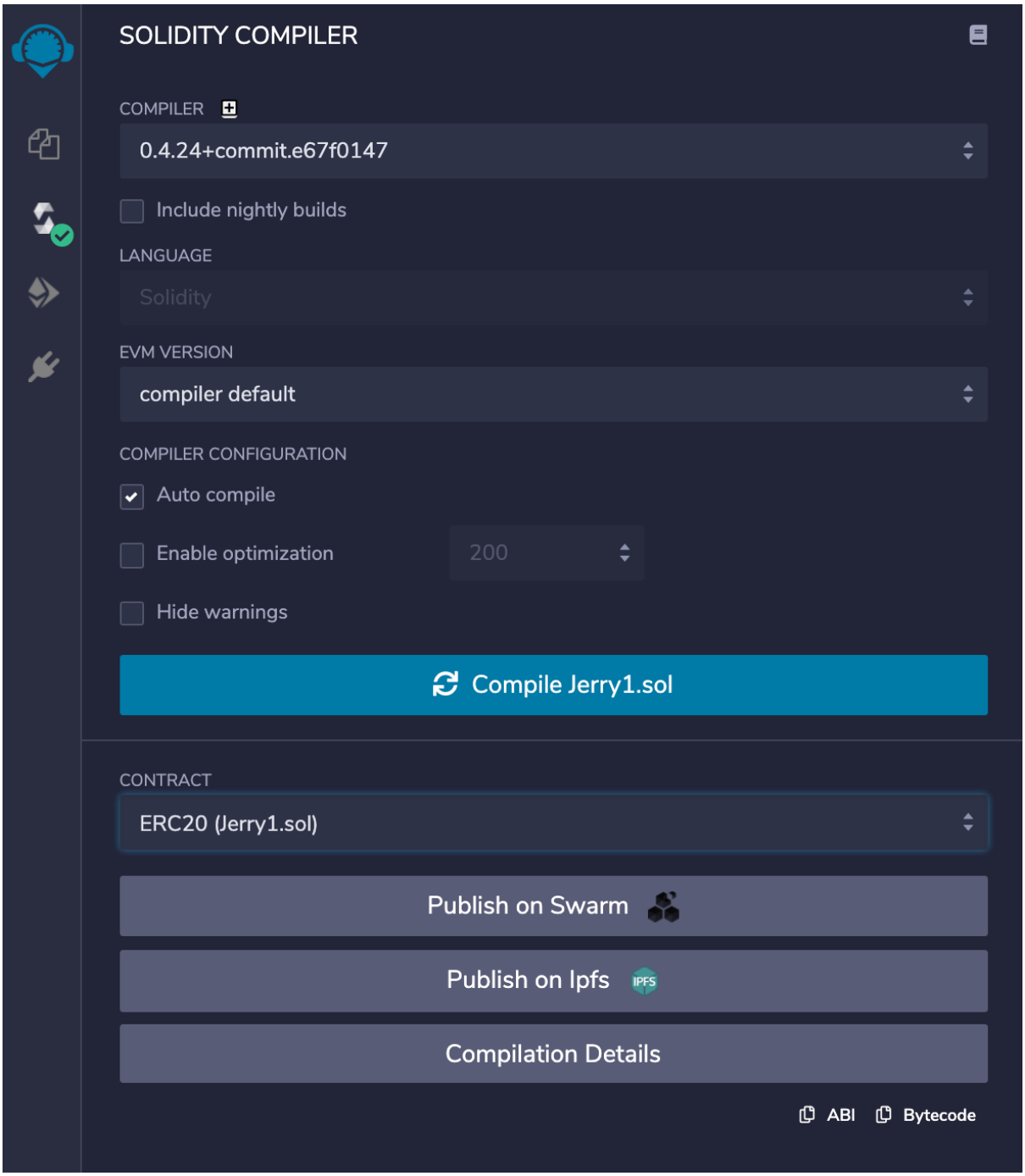
In detail:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Automated Audit

Remix Compiler Warnings

It throws warnings by Solidity's compiler. If it encounters any errors the contract cannot be compiled and deployed.



Disclaimer

Block Audit is not a security warranty, investment advice, or an endorsement of the Jerry contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

- Vulnerability analyzer built by BAR that quickly searches within smart contracts for vulnerable code fragments not only exact but also syntactically different clones.
- Sound formal verifier built by BAR that analyzes within smart contracts for proving that a program satisfies nonexistence of vulnerability such as integer overflow.
- Symbolic analyzer built by BAR that extracts code patterns within smart contracts for finding syntactic and semantic bugs and vulnerabilities based on SWC specification.

BLOCK AUDIT REPORT

Official Website
www.blockaudit.report

E-Mail
team@blockaudit.report

Twitter
@blockauditreport_team

Github
https://github.com/blockauditreport