Coverity Scan Static Analysis Report

**T1** 2023

Statement of Intent

Overview

This document aims to provide a record of static code analysis performed on a specific issue from the Coverity SAST scan for the NASA ION Open-Source code 4.1.1 project.

The primary purpose of this document is to validate the issue identified via the automated detection process to eliminate false positives.

Depending on findings, secondary purposes can include but are not limited to listing/providing recommended fixes alongside a list of attack vectors and potential exploits for consideration.

Reporting Best Practices

Please ensure best practices are kept when completing the document via regularly updating the Acronyms and Abbreviations table alongside any iterations made to the Document History table. This will allow other members to identify any updates and progress made across trimesters easily.

When using code snippets, please use screenshots that are clear and easy to read, alternatively, use words built-in code formatted found [here](https://appsource.microsoft.com/en-us/product/office/WA104382008?tab=Overview).

Document Naming Conventions

Naming conventions for this file are as follow; SAR\_{CID}. For example, when investigating issue 123456 the file name would be SAR\_123456.docx

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Dates** | **Version** | **Author** | **Comments** |
| 23/03/2023 | 1.0 | Connie Cox | Initial document. |
|  |  |  |  |
|  |  |  |  |

Table of Content

Contents

[Introduction 3](#_Toc119848724)

[Objective 3](#_Toc119848725)

[Scope 3](#_Toc119848726)

[Acronyms and Abbreviations 3](#_Toc119848727)

[Code Review and Analysis 4](#_Toc119848728)

[Outcomes 4](#_Toc119848729)

[Observations 4](#_Toc119848730)

[Supporting Evidence 4](#_Toc119848731)

[Conclusions and Recommendations 4](#_Toc119848732)

[References 5](#_Toc119848733)

[Appendix 6](#_Toc119848734)

# Introduction

## Objective

The primary objective of this analysis is to determine whether the defects identified in the Coverity Report for the ION Open Source 4.1.1 project are:

* Indeed, defects.
* Potentially exploitable.

The secondary objective of this analysis, where applicable, is to provide the following:

* Recommendation(s) to fix.
* Any exploit for consideration.

## Scope

This static code analysis is limited to the **High Impact Quality** type defect identified in the following CID:  
**1520894**

# Acronyms and Abbreviations

Please keep an updated list of acronyms and abbreviations used throughout the report.

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| DTN | Delay/Disruption Tolerant Network |
| ION | Interplanetary Overlay Network |
|  |  |

# Code Review and Analysis

## Introduction

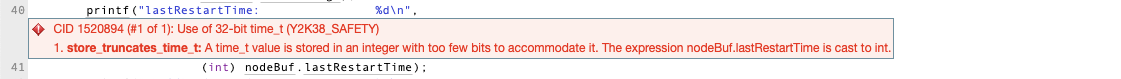
In analysing bpnmtest.c code base within the Bundle Protocol v7 test directory [/bpv7/test], Coverity highlighted a “high impact quality” vulnerability with the use of casting INT to the time\_t data type at line 41 in the int main (int argc, char \*\*argv) function .

The main function is the program’s entry point to bpnmtest.c whereby, it retrieves the address of the bundle node requested and the disposition of that node. The focus of this report is on what the main function executes with respect to the node that it retrieves.

## Observations

Within the int main (int argc, char \*\*argv) function, a call to bpnm\_node\_get() function is made to get the details of the node and print the details to screen.

A printf() function at line 40 displays the last restart time (nodeBuf.lastRestartTime) of the node as an integer; i.e. it casts an integer to the last restart time of the node before displaying the value to the screen. In doing so, Coverity is calling this out as being an issue as the time\_t value is being stored in an integer with too few bits to accommodate the time\_t value.



### Casting Smaller Primitives

Casting a smaller primitive to a larger primitive (time\_t to int) means that the program is converting the value held in time\_t data type to an integer data type causing the value in time\_t to be truncated if it’s larger than what can be held in an integer variable.

## Supporting Evidence

It needs to be noted that the time\_t datum represents a time value whereas upon conversion, the integer variable will hold a truncated value representing the number of seconds since the Unix epoch. The integer value simply holds a numerical value that does not necessarily correspond to a valid time value.

From a memory allocation perspective, depending on the target platform and the implementation of C, an integer data type is at least 2 bytes (16 bits) in size and can hold values between -32,768 to 32,767. A time\_t data type is at least 4 bytes (32 bits) in size and can hold a time value representing the number of seconds since the Unix epoch (January 1, 1970, 00:00:00 UTC) which typically equates to 32-bit signed integer range of -2,147,483,648 to 2,147,483,647 corresponding to the date values from 1901-12-13 20:45:52 UTC to 2038-01-19 03:14:07 UTC.

# Conclusions and Recommendations

This is not an exploitable vulnerability. The issue can be rectified by using functions such as gmtime() and use other time representations such as struct tm or struct timeval.

References  
Please keep an updated references list in APA7; The Deakin referencing guide can be found [here](https://www.deakin.edu.au/__data/assets/pdf_file/0009/2236752/Deakin-guide-to-APA7.pdf).

Appendix

Include additional information/documentation here to help the readers understand complex information.