**T2** 2023

Coverity Scan Static Analysis Report

Hardhard Enterprises

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 formatter 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** |
| 4/9/23 | V1.0 | Dean Scanlon | 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 5](#_Toc119848732)

[References 8](#_Toc119848733)

[Appendix 9](#_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 ***API Usage errors*** type defect identified in the following CIDs:  
***1520857 & 1520865***

# 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

## Overview

This issue occurs in 2 CID’s as identified by Coverity Statoc Analysis Tool.

Instance one occurs in the *bpcp.c* code segment within the *cfdp/utils/* directory, and instance 2 occurs in the *ui\_input.c* code segment within the *nm/mgr/* of NASA ION Delay Tolerant Network protocol.

Coverity Static Analysis Tool was able to detect **Medium Impact Quality** vulnerabilities that relate to an **Invalid type in argument to printf format specifier**.

## Observations

The bpcp.c code segment is part of the CFDP (CCSDS File delivery protocol) within the ION DTN protocol and is responsible for the delivery of files over networks that are subject to disruptions such as those in operation in space networks.

The error which is flagged by Coverity occurs on Line 1560 of the code where the code is using ‘%i’ format specifier within the PRINTF\_ARGS function.

The TID22 argument which is passed to the dbgprintf function is erroneous as it is of type ‘unsigned long int’ and the function is expecting type ‘int’.

In c language, format specifier ‘%d’ takes the base of the value as decimal, while ‘%i’ takes the base of the value as either decimal, octal or hexadecimal, depending on the value entered. Any value that begins with ‘0x’ is treated as hexadecimal, ‘0’ as octal, otherwise the value will be treated as decimal.

In the second instance of this report, CD\_1520865, Coverity alerts that on line 229, function vfprintf is being utilized which uses a printf-style format string. In this case, data is passed to the PRINTF\_ARGS once again which is of type unsigned long when an integer was expected.

Under certain circumstances, format strings can be exploited. If a user input is required by a program, this could lead to the insertion of format specifiers which points the printf function to a certain physical memory location which could lead to a variety of attacks. In this case however, the errors occur in a section of code which does not require/allow user input and therefore the errors should be considered as bugs rather than vulnerabilities.

## Supporting Evidence A computer screen shot of a computer code Description automatically generated

**Fig1. Coverity CID\_1520857**

A screenshot of a computer

Description automatically generated

**Fig2. Coverity CID\_1520857**

A screen shot of a computer screen

Description automatically generated

**Fig3. Coverity CID\_1520865**

A screenshot of a computer

Description automatically generated

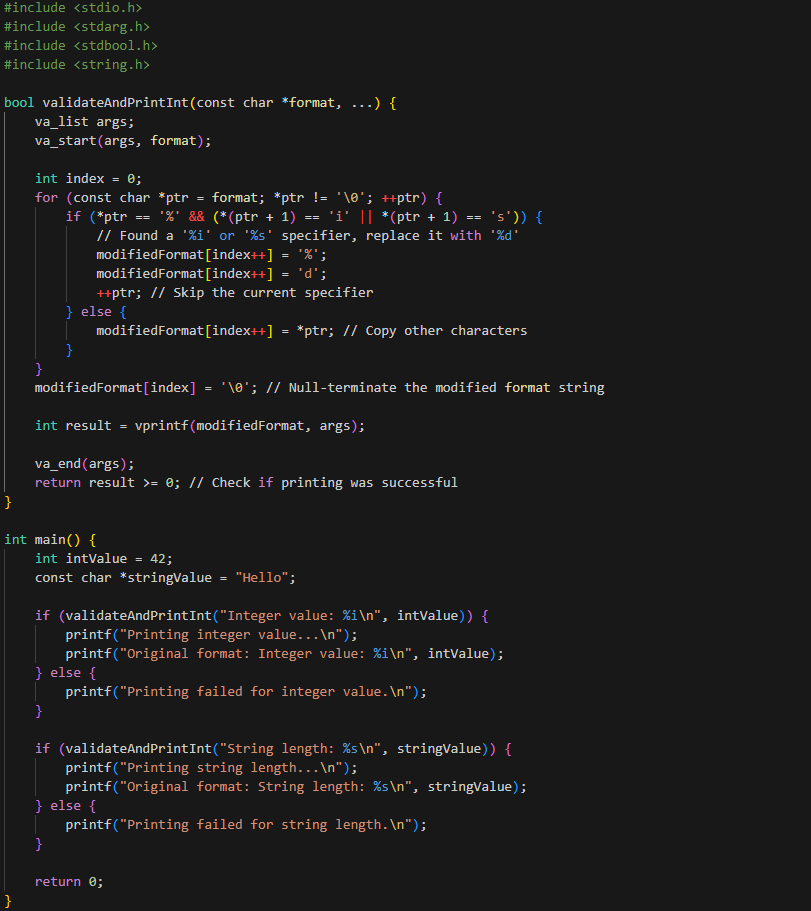
**Fig4. Coverity CID\_1520865**

# Conclusions and Recommendations

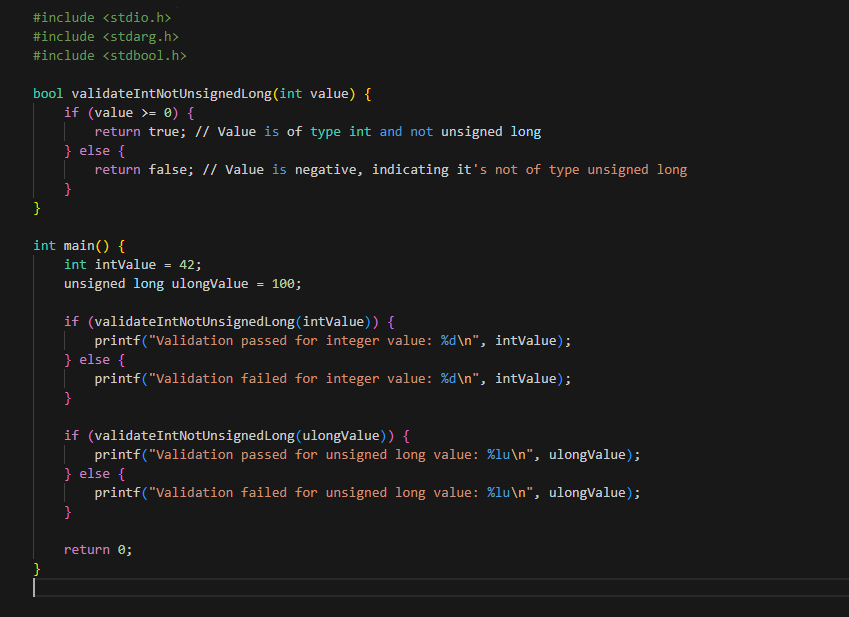
These errors occur essentially because data types are not being specified when being passed to the PRINTF\_ARGS function which handles printf outputs.

To correct this error, proper type definitions need to be implemented in the code before the data is passed to the printf function to ensure correctness of output.

Alternately, a validation function could be created to verify that data is in the correct format when being handled by the printf function. An example of a validation function which converts data that has format specifier %i and %s to integer type is shown below in Fig5. An example of a validation function to verify whether a data type is int or unsigned long is included in Fig6. These examples are out of context and are for demonstrative purposes only.



**Fig3. An example of a validation function for checking format specifier and converting to %d format specifier.**



**Fig3. An example of a validation function for checking data type**

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.