Coverity Scan Static Analysis Report

Hardhard Enterprises

**T3** 2022

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** |
| 5/12/2022 | 1.0 | Jesse Ludeman | Initial document |
| 6/12/2022 | 1.1 | Jesse Ludeman | Update observations and exploit |

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 4](#_Toc119848733)

[Appendix 4](#_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 ***Uninitialized scalar variable*** type defect identified in the following CIDs: 1520776

# 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

## Outcomes

When performing static code analysis using the ION Open Source 4.1.1 dashboard for CID 1520776, there is a high impact problem that involves an uninitialized scalar variable in the main method of the /bpv7/test/bpnmtest.c file.

Stack variables in the C and C++ programming language are not initialized by default, and generally contain “junk” data before the function is invoked. The problem here is that an attacker can potentially exploit this weakness by reading and writing these variables prior to the function being invoked.

## Observations

This type of error occurs when a variable has not been initialized correctly, which can subsequently lead to unpredictable or undesirable results.

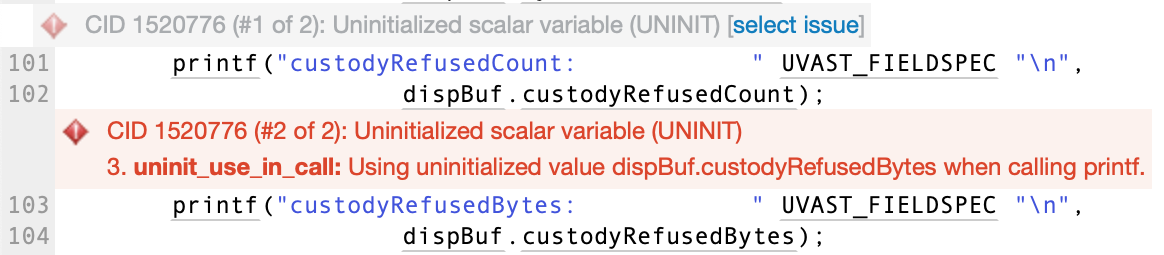
  
Figure 1 – First occurrence of this error

Figure 1 demonstrates where Coverity has detected the offending code. The printf function is being used to print the results of custodyRefusedBytes from the dispBuf struct.

|  |
| --- |
| typedef struct {  uvast custodyRefusedCount;  uvast custodyRefusedBytes;  ...;  ...;  }  NmbpDisposition; |

Figure 2 – The struct where custodyRefusedBytes originates

Figure 2 shows how the NmbpDisposition struct declares the custodyRefusedBytes variable. The uvast variable is an unsigned long data structure. Other variables have been redacted for readability.

## Supporting Evidence

It is not considered best practice to declare uninitialized variables in C / C++, as this can result in an undesirable outcome during program execution. Furthermore, uninitialized variables can be used by an attacker to exploit a program.

If an attacker knows the datatype of the uvast variable (which is an unsigned long), then this could potentially be initialized to the maximum that this datatype supports. For example, an unsigned long can be assigned a value in the range of [ 0, 4,294,967,295 ]. Setting this value to the maximum would not only initialize the variable but could potentially cause undesirable program execution issues when the datatype is referenced. This has been demonstrated in figure 3.

|  |
| --- |
| int main(int *arcg*, char \*\**argv*)  {  *NmbpDisposition* dispBuf = {  .custodyRefusedBytes = 4294967295};  } |

Figure 3 – Initializing custodyRefusedBytes to the maximum for an unsigned long

# Conclusions and Recommendations

To mitigate this problem, initialize the custodyRefusedBytes variable prior calling it and using it further down in the main function.

|  |
| --- |
| int main(int *arcg*, char \*\**argv*)  {  *NmbpDisposition* dispBuf = {  .custodyRefusedBytes = 5};  } |

Figure 4 – Recommended fix to solve this vulnerability

Figure 3 demonstrates initializing custodyRefusedBytes to 5. This effectively prevents an attacker from reading and writing to this variable prior to program’s execution, as there is no “junk” data being assigned to it in the program stack. This variable can be set to a different value later if required.

References

Appendix

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