Coverity Scan Static Analysis Report

Hardhard Enterprises

**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 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** |
| 29/03/2023 | V0.1 | Callam | Initial investigation |
|  |  |  |  |
|  |  |  |  |

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 **Memory corruption** type defect identified in the following CIDs:  
**1520757**

# 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

This C code intends to remove a specific event from the event set (bsles\_clear\_event) and frees its memory. The function performs **sanity checks** then removes the event from the event mask, looks for the event object in the event set list of events. Once it finds this function the event is freed and the function frees its memory while the freed address is removed from the event set list.

## Observations

On line 155 the CHKERR macro checks to see if esPtr is NULL (see fig 1). If esPtr is NULL the macro returns -1 and the function closes. If it is not NULL the function will continue. In this case the condition is negated, meaning if esPtr is NULL the function continues and potentially causes a segmentation fault.

On lines 167-172, a loop iterates through the list of events in esPtr (see fig 2). elt is the assigned variable to hold the address of the current element in the list. The sm\_list\_next obtains the address of the next element in the list. If elt is the address of the last element in the list, the function returns **UINTMAX\_MAX** which is an **out of bounds** address. This causes an access violation when the memory is accessed at that address.

Lines 166-168 display loop traversing through a list of events (see fig 3). It is noted that there is no check for the end of the list. The loop should exit when elt is equal to 0 (end of list has been reached) but it does not. As a result sm\_list\_next() is called with elt as the argument, returning UINTMAX\_MAX (18446744073709551615) which is an out of bounds address which causes the out of bounds access. The result of sm\_list\_next() is assigned to elt, if elt is assigned UINTMAX\_MAX the loop would not not exit as expected and elt would be UINTMAX\_MAX for the remainder of the loop.

## Supporting Evidence



Figure 1: Null check on esPtr

# 

# Graphical user interface, text, application, email Description automatically generated

Figure 2: Loop through event list in esPtr

# Graphical user interface, text Description automatically generated with medium confidence

Figure 3: Loop iterates through a list of events

# Conclusions and Recommendations

Add code in lines 166-168 so the loop ends when elt=0, so the loop will terminate at the end of the list avoiding the unsigned integer.

Add checks to ensure the elt argument is valid before calling sm\_list\_data function such as checking if elt = 0

Add code to return a value of -1 to indicate failure instead of the current code which can return 1 (successful function) even if the branch is false, so eventID (line 157) does not proceed with the function since it must be positive

This code has the potential to be exploitable. Although out of bounds access does not cause memory leaks, it can lead to memory corruption which can eventually result in **memory leaks** if not handled correctly. This code has the potential to cause a memory overflow, in which the program attempts to write data beyond the boundaries of its specified memory block which can cause an overwrite of memory that was used for a different purpose, which in turn may cause program crashes, data corruption or the creation of security vulnerabilities.

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

**Out of bounds access –** When an array index is included and used when it is out of bounds, causing the compiler to run it, the output is typically incorrect.

**UINTMAX\_MAX –** Maximum value of unsigned integer type.

**Memory Leaks –** When a program allocates memory for a certain task but fails to release that memory when the task is completed leading to an accumulation of unreleased memory causing the program to slow down or crash.

**Sanity Checks –** A basic check or test to evaluate whether code is stable enough to proceed with further testing.