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** |
| 19/11/2022 | 0.1 | Connie Cox | Initial Investigation |
| 23/11/2022 | 0.2 | Connie Cox | Minor updates to CID number, acronyms and abreviations section, and Observations sections. |
|  |  |  |  |

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 ***Out-of-bounds access*** type defect identified in the following CIDs:  
**1520890**

# Acronyms and Abbreviations

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

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| CID | Coverity Issue Identification Number |
| DTN | Delay/Disruption Tolerant Network |
| ION | Interplanetary Overlay Network |
|  |  |

# Code Review and Analysis

## Outcomes

Please provide an in-depth report on the analysis performed here.

To understand the defect outlined in the Coverity report, an understanding of the findSpan() function was undertaken. The function is outlined as follows:

1. void findSpan(uvast engineId, LtpVspan \*\*vspan, PsmAddress \*vspanElt)
2. {
3. PsmPartition ltpwm = getIonwm();
4. PsmAddress elt;
6. CHKVOID(ionLocked());
7. CHKVOID(vspan);
8. CHKVOID(vspanElt);
9. for (elt = sm\_list\_first(ltpwm, (\_ltpvdb(NULL))->spans); elt;
10. elt = sm\_list\_next(ltpwm, elt))
11. {
12. \*vspan = (LtpVspan \*) psp(ltpwm, sm\_list\_data(ltpwm, elt));
13. if ((\*vspan)->engineId == engineId)
14. {
15. break;
16. }
17. }
19. \*vspanElt = elt; /\* (Zero if vspan was not found.) \*/
20. }

This function using the index values contained in the doubly linked list and iterates through the doubly linked list to retrieve the PsmAddress of the span with the engineID that matches the engineID in the parameter. If the iteration finds a match, it assigns the address pointer of that span element to pointer variable vspanElt. If no match is found, zero is assigned.

The linked list functions sm\_list\_first() returns the PsmAddress of first element in the list, sm\_list\_next() returns the PsmAddress of the next element in the list and sm\_list\_data() returns the PsmAddress of the data value for the element.

It is through the iteration of doubly linked list holding the span addresses that Coverity identifies an issue. The highlighted section in the code above is the section of code that is in question.

## Observations

Please provide any observations, and feel free to make references to documents in the appendix.

Whilst there are checks in place for void values for if ION is locked, vspan and vspanElt; there’s no validation on the index values returned in sm\_list\_first() and sm\_list\_next() or used in sm\_list\_data(). There is also no validation on the value returned from the getIonwm() function and assigned to the PsmPartition ltpwn. According to Coverity, the index value is suspicious because it is larger than expected. Possible root cause in this case is that the parameter provided was negative and interpreted as unsigned. This could then cause an Out-of-bounds access (OVERRUN) defect when calling sm\_list\_data().

### What is an out-of-bounds access defect?

An out-of-bounds access error occurs when the linked list accesses memory and/or stores data beyond its original memory allocation. In this context, the index used in the linked list storing PsmAddresses for span elements could potentially be larger than the max index that was allocated to the original linked list. This has the potential to:

* Cause adjacent storage to store overflowed data
* Crash the node(s)
* Create an entry point for a cyber exploit

## Supporting Evidence

Please provide any supporting evidence, and feel free to make references to documents in the appendix.

# Conclusions and Recommendations

Please provide any conclusions and recommendations here. Include references to supporting evidence and any other required information in the appendix.

To ensure that buffers are not over run, there needs to be bounds checking on the index being used. This can be done by verifying the parameter used. That is, if the parameter is negative, it should not be treated as unsigned. Secondly, validation of the index returned. The index should never be larger than the number of span elements in the configuration. To streamline this process, a function similar to VOIDCHK() could be created to perform the index value validation.

Whilst this analysis was specific to CID 1520890, the conclusion could apply to all Out-of-bounds access issues identified in the Coverity report, especially if the issue arises when using linked lists to retrieve PsmAddresses of spans.

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.