tre

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 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** |
| 12/05/2023 | V1.0 | Jesse Ludeman | Initial document and investigation |
| 15/03/2023 | V1.1 | Jesse Ludeman | Finalize 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 ***Out-of-bounds access*** type defect identified in the following CIDs: 1520800

# 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

CID 1520800 has been flagged by Coverity as a High impact issue of type *Out-of-bounds access*. This type of defect typically indicates that the product is conducting actions on a memory buffer, but it has the capability to access or modify a memory location that extends beyond the intended limits of the buffer. This has been flagged in the ltpStop() function, which according to the documentation and my research, is the reverse of the ltpStart() function. This function sits within the Licklider Transmission Protocol, which is a point-to-point protocol for use in deep space links.

## Observations

The first errors flagged initially occur on line 1270, which advise us that the function sm\_list\_next() may return value 18446744073709551615. This occurs because inside the for loop, the elt loop variable is being assigned the value from the sm\_list\_first() function. Additionally, inside the body of the for loop, we notice that the elt variable is then reassigned the value from the sm\_list\_next() function during each iteration.

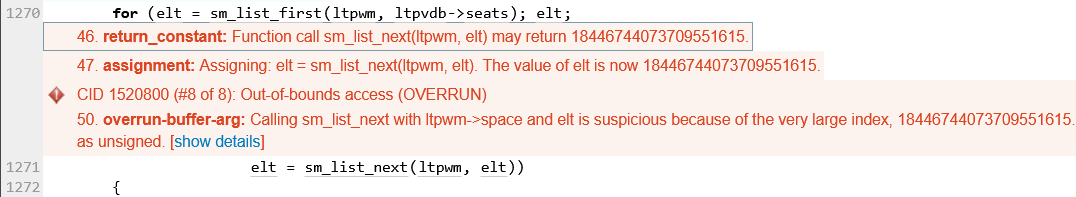


Figure 1 - The initial error

(Manpages 2014: Description) The API documentation for the sm\_list\_next() function indicates that the second parameter is a PsmAddress. Because the elt variable has been assigned the value 18446744073709551615, Coverity flags this as an error due to the large index.

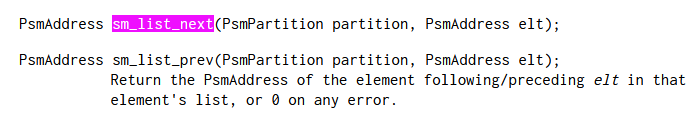


Figure 2 - sm\_list\_next() function

## Supporting Evidence

# Conclusions and Recommendations

Observing the sm\_list\_next() function works as intended as noted in Figure 3. However, Hardhat Enterprises recommend that it be updated to include validation on the elt variable. For example, if PsmAddress value has a known size, then this should be considered and added as a conditional check inside the function, this has been demonstrated in Figure 4. In a scenario where the elt variable is assigned a large value, this would prevent it from being used unintentionally in the application.

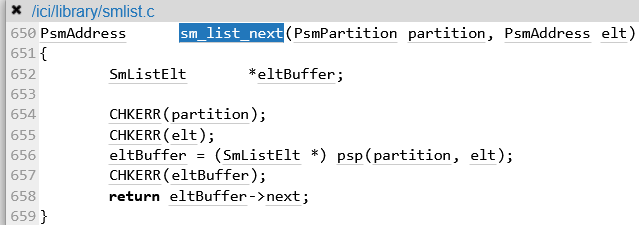


Figure 3 sm\_list\_next() function1

|  |
| --- |
| PsmAddress sm\_list\_next(PsmPartition partition, PsmAddress elt)  {      if (elt > 10000)      {          printf("PsmAddress value too great");          return -1;      }      SmListElt \*eltBuffer;      CHKERR(partition);      CHKERR(elt);      eltBuffer = (SmListElt \*)psp(partition, elt);      CHKERR(eltBuffer);      return eltBuffer->next;  } |

Figure 4 - Example fix provided

Whilst no security vulnerability has been identified for this defect, it’s important to realize that *Out-of-bounds access* errors have the potential to be exploitable. For example, the buffer overflow and denial-of-service attacks are possible for this type of issue. However, given the context of the existing codebase, we are not able to identify any potential risks.

References  
Debian Manpages (2014) [smlist(3) – ion.doc – Debian jessie – Debian Manpages](https://manpages.debian.org/jessie/ion-doc/smlist.3.en.html), accessed 12 May 2023.

Appendix

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