**T1** 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** |
| 28/03/2023 | V0.1 | Anthony Scantsonihas | Investigation |
| 02/04/2023 | V1.0 | Anthony Scantsonihas | Report Finalization |

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:

***CID1520688***

# 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 |
| CID | Coverity Issue Identification Number |
| CWE | Common Weakness Enumeration |

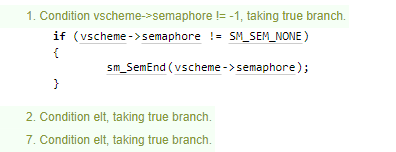
# Code Review and Analysis

## Overview

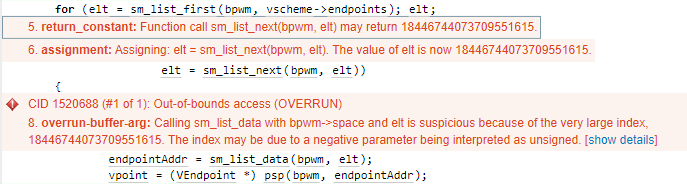
The Coverity report for the CID 1520688 has discovered issues found within the code base libbP.c:706. The issue found for this segment of code is Out-of-bounds access which is under CWE-119. The error is also commonly known as a Buffer Overflow which typically indicates that the code is reading and writing to a memory location from the outside of the intended boundary of the buffer.

## Observations

The issue can be traced to the stopScheme function which is responsible for stopping a VScheme by ending the associated semaphore and therefore ending the forwarder task.



The if statement checks to see if the value of the “vscheme->semaphore” variable is not -1, if it is true the function sm\_SemEnd() gets called which is used to end the sephamore associated with the VScheme. This could be a potential vulnerability given that if the “vscheme->semaphore” variable is set to -1 it could stop the code from executing correctly.



There also appears to be an issue with the “sm\_list\_next” function as it returns an out-of-bounds address which then sets off the overrun buffer error when passed into the function “sm\_list\_data”.

**-Potential Vulnerabilities**

It is also possible that an attacker can use this error within the code to exploit the program by performing a denial of service attack by forcing the code into an infinite loop and therefore overflowing it with data, and preventing the program from executing correctly.

## Supporting Evidence

Evidence of the Buffer Flow issues can be seen in the Coverity error flag seen below. The error flag also addresses how the large index created in the bufferflow error is most likely caused by one of the parameters being set to a negative value and therefore causing the code to read it as being unassigned. This means that an attacker could intentionally set that variable to be a negative number and induce the buffer overflow error to crash the program and prevent it from running correctly.



# Conclusions and Recommendations

The recommendations I would provide to address this issue would be to implement code that validates the index to catch any potential errors that may occur during the runtime of the program. With this if an attacker tries to change the variables maliciously to induce a denial of service attack, the code will be able to identify it and prevent it from causing the program to crash.

References

MITRE Corporation. (2023, January 31). CWE - CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer. Retrieved March 21, 2023, from <https://cwe.mitre.org/data/definitions/119.html>

Appendix

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