**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** |
| 19/05/2023 | V1.0 | Adharshaan Devaraj | Initial Document |
|  |  |  |  |
|  |  |  |  |

Table of Content

Contents

[Introduction 3](#_Toc133624419)

[Objective 3](#_Toc133624420)

[Scope 3](#_Toc133624421)

[Acronyms and Abbreviations 3](#_Toc133624422)

[Code Review and Analysis 4](#_Toc133624423)

[Overview 4](#_Toc133624424)

[Observations 4](#_Toc133624425)

[Supporting Evidence 4](#_Toc133624426)

[Conclusions and Recommendations 4](#_Toc133624427)

[References 5](#_Toc133624428)

[Appendix 5](#_Toc133624429)

# 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 – corruptions*** type defect identified in the following CIDs:  
***1520721***

# 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

## Overview

The **Linux Test Project suite (LTP)** contains the “**libsda.c**” file, which contains the Out of Bounds access error. The “**ltp/sda**” library provides functions and utilities for initializing data acquisition devices, establishing acquisition parameters (such as sample rate and data format), initiating, and preventing data acquisition, retrieving acquired data, and other associated activities. The code for these functions, as well as their implementation details, would be stored in the "**libsda.c**" file.

The issue is also known as a Buffer Overflow, and it often implies that the code is reading and writing to a memory address from outside the buffer's intended boundary.

## Observations

According to the error flag, one of the parameters was most likely assigned a negative value, which the code viewed as unassigned. This led in a buffer overflow error, which resulted in a massive index. If a hacker modifies the variable to a negative value on purpose, it might trigger a buffer overflow, causing the program to crash and cease working correctly.

## The issue that has identified is that the **sdr->dssm** parameter has a very big value, 18446744073709551615. Because this is the highest value that can be expressed using 64 bits, it's conceivable that it was allocated to **sdr->dssm** as a consequence of a mistake or unusual situation. If this is the case, invoking the **zco\_destroy** () method with this parameter may result in problems, such as an attempt to access memory outside of the ZCO object's allocated range.

## Supporting Evidence

A picture containing text, font, screenshot

Description automatically generated

Two effects of buffer overflow vulnerability:

* Buffer overflows can be exploited to execute arbitrary code that is not covered by the implicit security policy of a program.
* Buffer overflows also cause crashes, which cause a lack of availability and send the application into an unending loop.

Given the attacker's knowledge of the source code and the issues that emerge when computations of this function are activated, this error must be rectified promptly. (CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer, 2023)

# Conclusions and Recommendations

In general, if you suspect that your organization is exposed to the OVERRUN vulnerability, you should act as quickly as possible. This may entail doing a full security evaluation of your systems, deploying any applicable patches or upgrades, and putting in place extra security measures to prevent unwanted access. It may also be a good idea to seek the advice and support of cybersecurity professionals.

One of the most effective ways to prevent out-of-boundaries accesses and potential security vulnerabilities is to ensure that array subscripts remain inside the array's legal bounds and that the array is not accessed outside of its legitimate memory space. This can assist in ensuring that the software only accesses data inside the allotted memory space and preventing attempts to access sensitive data or crash the system. In addition to validating array subscripts, it is critical to apply safe coding techniques and to evaluate and update the systems on a regular basis to ensure that they are protected against known vulnerabilities.

To ensure that an array is not accessed beyond its valid memory area, you can take the following steps:

* Verify that the array subscripts used in your code are within the legal bounds of the array. This means that the subscripts should be greater than or equal to 0 and less than the size of the array.
* Use appropriate data types for the array subscripts. For example, if the size of the array is stored in a variable of type int, the array subscripts should also be of type int to ensure that they can represent the full range of legal values.

The argument should not be considered as unsigned if it is negative. The element index returned should be validated. The element index should never be more than the number of configuration bundle elements. To make this procedure more efficient, an element index value validation function might be written.

# References

*CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer*. (2023). Retrieved from Common Weakness Enumeration: https://cwe.mitre.org/data/definitions/119.html

Appendix

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