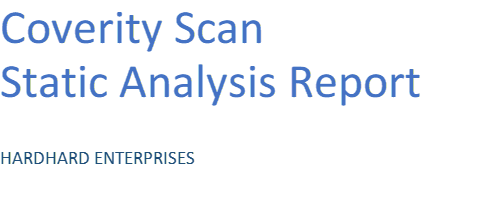
​



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** |
| 8/12/2022 | V0.1 | Yongxinag Huang/Yizhou Feng | Initial investigation |
| 11/12/2022 | V0.2 | Yongxinag Huang/Yizhou Feng | Finish investigation |

Table of Content

Contents

 Introduction ……………………………………………………………………………………………………………………… 2

Objective ……………………………………………………………………………………………………………………………2

Scope ………………………………………………………………………………………………………………………………2

Acronyms and Abbreviations ……………………………………………………………………………………………………2

Code Review and Analysis ………………………………………………………………………………………………………. 3

Outcomes ………………………………………………………………………………………………………………………… 3

Observations ………………………………………………………………………………………………………………………3

Supporting Evidence ………………………………………………………………………………………………………….. 4

Conclusions and Recommendations ………………………………………………………………………………………… 4

References …………………………………………………………………………………………………………………………… 5

Appendix ………………………………………………………………………………………………………………………………. 5

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 ***Resource leak*** type defect identified in the following CIDs: 1520891.

Acronyms and Abbreviations

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

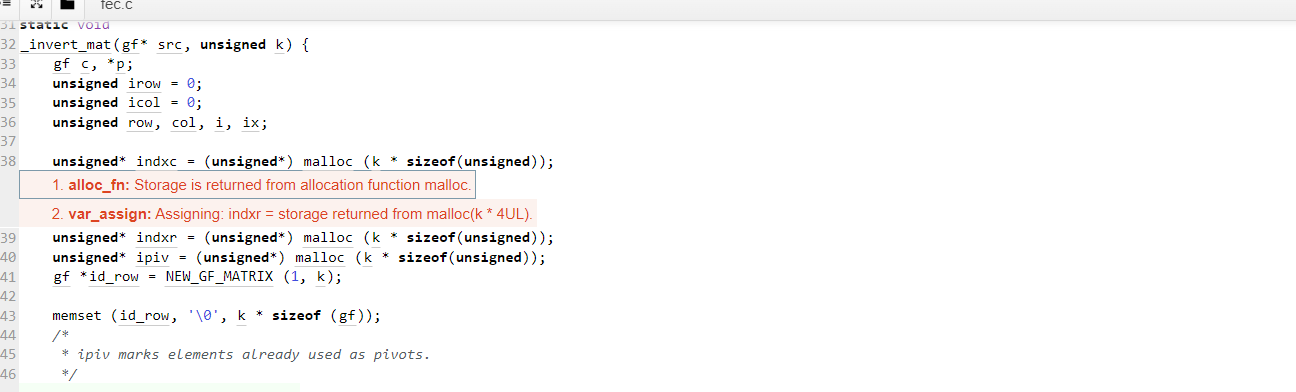
|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| DTN | Delay/Disruption Tolerant Network |
| ION | Interplanetary Overlay Network |
| CID | Coverity Issue Identification Number |

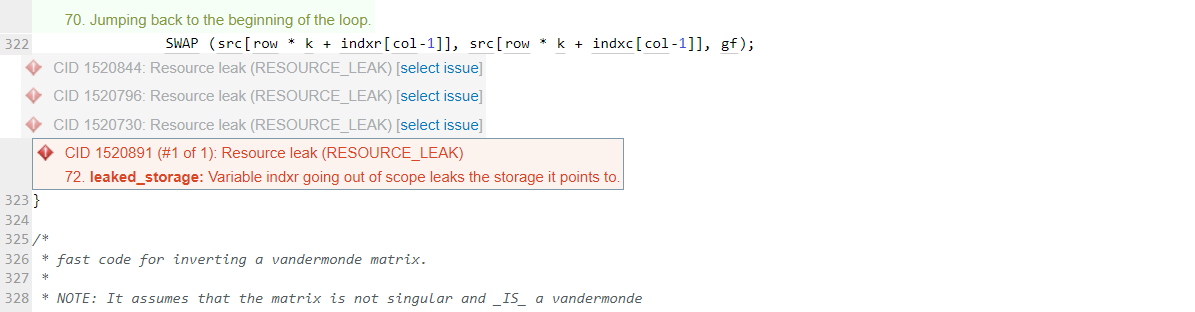
Code Review and Analysis

Outcomes

When I performed static code analysis on CID 1520891 using the ION Open Source 4.1.1 dashboard, I found that CID1520888 had a high impact issue involving Resource Leak. The Coverity Scanner pointed out that in line 239. alloc\_fn: Storage is returned from allocation function malloc. In line 239. var\_assign: Assigning: indxr = storage returned from malloc(k \* 4UL). in line 323: sleaked\_storage: Variable indxr going out of scope leaks the storage it points to.

## Observations





The error message alloc\_fn: Storage is returned from allocation function malloc indicates that the malloc function, which is used to allocate memory dynamically at runtime, has returned a null pointer. This typically indicates that there is not enough memory available to allocate the requested amount of memory.

In the code snippet you provided, the malloc function is being used on lines 238, 239, and 240 to allocate memory for three arrays: indxc, indxr, and ipiv. The size of each array is determined by the k parameter, which is passed to the 232\_invert\_mat function when it is called.

The error message var\_assign: Assigning: indxr = storage returned from malloc(k \* 4UL) indicates that the malloc function, which is used to allocate memory dynamically at runtime, has returned a null pointer. This typically indicates that there is not enough memory available to allocate the requested amount of memory.

In the code snippet you provided, the malloc function is being called on line 238 to allocate memory for the indxr array. The size of the array is determined by parameter k, which is passed to the 232\_invert\_mat function when it is called.

## Supporting Evidence

What is resource leak:

A memory leak in C is a specific kind of resource leak that happens when a computer software or program poorly handles memory allocations and fails to free up the memory that is no longer required. A memory leak can also occur when a variable/object is kept in memory but is not accessible to the program that is running.

As there is limited memory available in each system, memory leaks can have an effect on lowering the system's performance. In the worst-case scenario, too much of the available memory may be allocated, causing the system or device to malfunction entirely or partially, the programs may crash, or the system may significantly slow down.

What may cause resource leak:

A memory leak in C typically happens when the pointer loses its initial allocated address due to which the pointed memory block cannot be accessed or cannot de-allocated, and this becomes the reason for the memory leaks.

There are other reasons may cause resource leak:

1. Assigning a different address value to the pointer variable before de-allocating the memory block already pointed by the pointer.

2.If a pointer variable goes out of scope, it may lead to a memory leak.

3.Not deallocating the memory before the program execution finishes.

4.If an error occurred during the de-allocation of the memory block.

5.Visit the following link to know more about dynamic allocation and de-allocation of memory blocks using malloc() and calloc() in C Language, Dynamic Memory Allocation in C.

# Conclusions and Recommendations

A memory leak in C happens when a computer program poorly handles memory allocations and fails to free up the memory that is no longer required. Memory leaks can have an effect on lowering the system's performance.

Here’s someway to avoid resource leak:

1. Use a free() function statement associated with every malloc() or calloc() function statement. For example:

char \*str = (char\*)malloc(n \* sizeof(char));

// code here

free(str);

Writing the free() statement immediately after the malloc() or calloc() function is a good practice. It helps to avoid the scenario when a programmer forgets to add a free() statement.

1. Avoid modifying the original pointer, use a temporary pointer. For example:

int \*ptr = (int\*)malloc(sizeof(int));

// making a copy of the pointer in the temp pointer variable

int \*temp = ptr;

free(ptr);

Working with a temporary pointer is a good practice since the original pointer keeps the address of the allocated memory, and it can get accidentally changed. So, if we work with a temp pointer, we can preserve the original pointer value and re-collect the address value to de-allocate the memory for avoiding any memory leaks.

References

Abhishek Chandra, n.d., 10 Nov 2022, ‘what is Memory Leak in C?’

Available at: <https://www.scaler.com/topics/memory-leak-in-c/>

Access on: 17/12/2022

Appendix