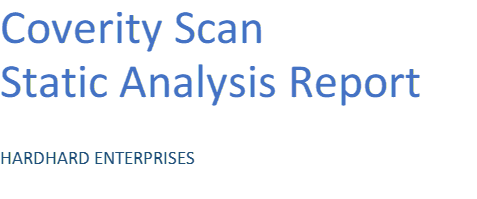
​



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** |
| 16/12/2022 | V0.1 | Yongxinag Huang, Yizhou Feng | Initial investigation |
| 17/12/2022 | V0.2 | Yongxinag Huang, Yizhou Feng | Simply updated background like Scope and Acronyms. |
| 18/12/2022 | V0.3 | Yongxinag Huang, Yizhou Feng | Updated Outcome and Observations. |

Table of Content

Contents

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

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

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

Acronyms and Abbreviations ……………………………………………………………………………………………………3

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

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

Observations ……………………………………………………………………………………………………………………4

Supporting Evidence ……………………………………………………………………………………………………….. 5

Conclusions and Recommendations ………………………………………………………………………………………… 6

References …………………………………………………………………………………………………………………………… 6

Appendix ………………………………………………………………………………………………………………………………. 7

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: 1520811

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 1520888 using the ION Open Source 4.1.1 dashboard, I found that CID1520888 had a high impact issue involving Out-of-Bounds access. The Coverity Scanner pointed out that in lines 501-502, the index is too large, possibly due to negative arguments being interpreted as unsigned.

This defect is contained in Sdr\_hash\_remove function. Which shows below:

 int     Sdr\_hash\_remove(**const** char \*file, int line, Sdr sdrv, Object hash,

                char \*key, Address \*value)

{

        int     keyLength;

        int     kvpairLength;

        int     rowSize;

        int     rowCount;

        int     rowNbr;

        Address rowAddr;

        Object  listAddr;

        Object  elt;

        Object  kvpairAddr;

        KvPair  kvpair;

        int     result;

**if** (!(sdr\_in\_xn(sdrv)))

        {

                oK(\_iEnd(file, line, \_notInXnMsg()));

**return** -1;

        }

        joinTrace(sdrv, file, line);

**if** (hash == 0 || key == NULL)

        {

                oK(\_xniEnd(file, line, \_apiErrMsg(), sdrv));

**return** -1;

        }

        keyLength = sdr\_table\_user\_data(sdrv, hash);

        kvpairLength = **sizeof**(Address) + keyLength;

        sdr\_table\_dimensions(sdrv, hash, &rowSize, &rowCount);

        rowNbr = computeRowNbr(rowCount, keyLength, key);

        rowAddr = sdr\_table\_row(sdrv, hash, rowNbr);

        sdr\_read(sdrv, (char \*) &listAddr, rowAddr, **sizeof**(Object));

**for** (elt = sdr\_list\_first(sdrv, listAddr); elt;

                       elt = sdr\_list\_next(sdrv, elt))

        {

                kvpairAddr = sdr\_list\_data(sdrv, elt);

                sdr\_read(sdrv, (char \*) &kvpair, kvpairAddr, kvpairLength);

                result = memcmp(kvpair.key, key, keyLength);

**if** (result < 0)

                {

**continue**;

                }

**if** (result > 0)

                {

**break**;  */\*      Not found.                      \*/*

                }

**if** (value)

               {

                        \*value = kvpair.value;

                }

                Sdr\_free(file, line, sdrv, kvpairAddr);

                Sdr\_list\_delete(file, line, sdrv, elt, NULL, NULL);

**return** 1;       */\*      Succeeded.                      \*/*

        }

**return** 0;               */\*      Unable to remove entry.         \*/*

}

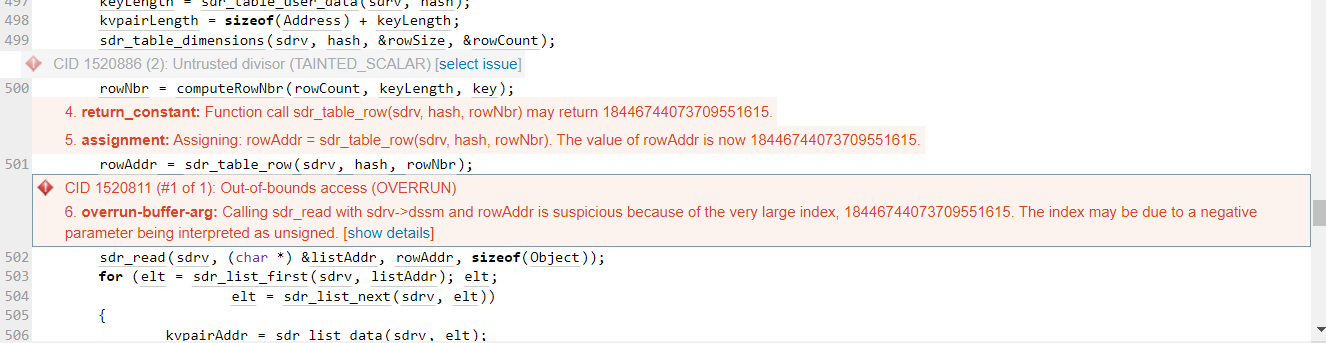
**Observations**

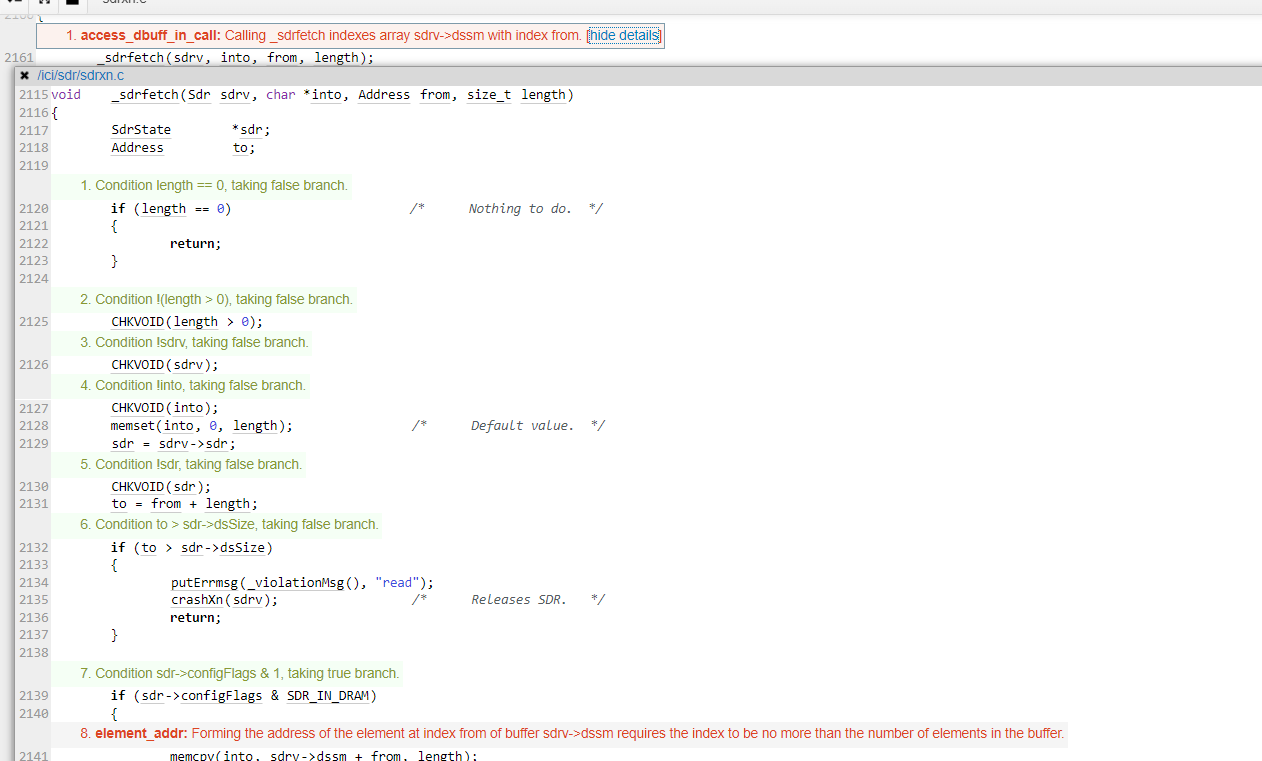
In my opinion, the function is lack of validation of the index value returned in sdr\_table\_row and the value used in sdr\_read. There’s also no validation on the value returned from the rowAddr function and assigned to the Sdr\_hash\_revise.

In general, though, the value 18446744073709551615 is a very large unsigned integer value, which is the maximum value that can be represented using 64 bits. It's possible that this value is being returned to indicate some kind of error or exceptional condition, but without more information about the function and its intended behavior, it's difficult to say for sure.

According to Coverity Scan, there’s some errors occur on the index values, Coverity Scan shows the index value is too large. The possible reason I think in this case is the parameter provided was negative, and was annotated as unsigned. I think this is how Out-Of-Bounds Access error occurs, because the Out-Of-Bounds Access error usually occur when a program attempts to access an index of array which is outside of its range.

**Supporting Evidence**





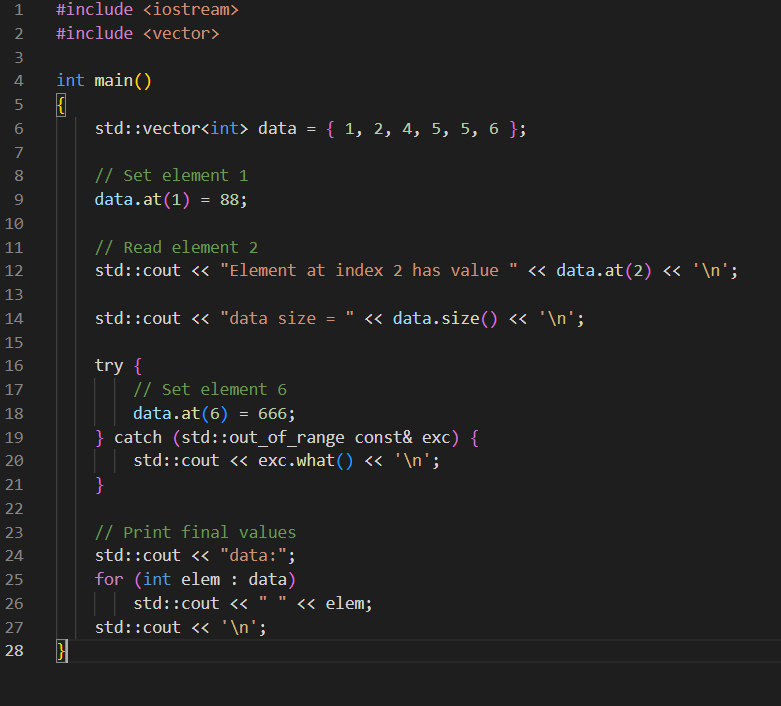
* What is Out-Of-Bounds Access:

The array index out of bounds error is a special case of the buffer overflow error. It occurs when the index used to address array items exceeds the allowed value. It's the area outside the array bounds which is being addressed, that's why this situation is considered a case of undefined behavior. [PVS-Studio, 2013]

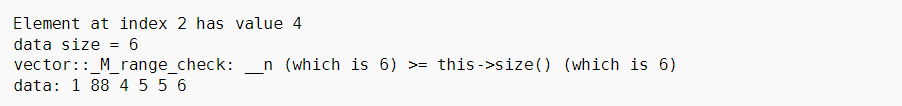
* How to solve Array Out-Of-Bounds Access problem in C/C++:

In C, we can do nothing but try to stay inside the bound of array, however, in C++,  C++ offers the std::vector class template, which does not require to perform bounds checking. A vector also has the std::at() member function which can perform bounds-checking. [geeksforgeeks.org, 2017]

Here’s a simply example to show how to use a std::at() member function to check the bounds:



OUTPUT:



We can see in the third line of the output, the range has been checked, we can use function like this to avoid array out-of-bounds access error. [cppreference.com, 2013]

**Conclusions and Recommendations**

In order to solve this high-impact problem, I think it should be checked whether the value of index is in the range, which is to check the parameter used. If the value of parameter is negative, it should not be marked as unsigned. And I think index returned also needs to be checked. I think this change can be achieved by creating a function.

**References**

PVS-Studio, n.d., 06 Mar 2013, Array index out of bounds [online]   
Available at: <https://pvs-studio.com/en/blog/terms/0071/>

Accessed: 03 December 2022

geeksforgeeks, n.d., 07 July 2017, Accessing array out of bounds in C/C++ [online]   
Available at: <https://www.geeksforgeeks.org/accessing-array-bounds-ccpp/>

Accessed: 03 December 2022

Cppreference.com, n.d., 31 May 2013, std::vector<T,Allocator>::at [online]   
Available at: <https://en.cppreference.com/w/cpp/container/vector/at>

Accessed: 03 December 2022

**Appendix**