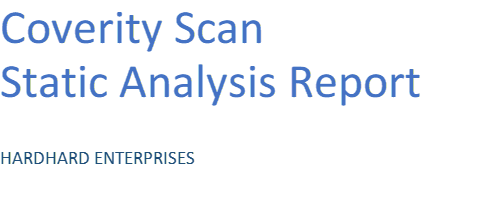
​



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** |
| 4/12/2022 | V0.1 | YIZHOU FENG/YONGXIANG HUANG | Initial document |
| 8/12/2022 | V0.2 | YIZHOU FENG/YONGXIANG HUANG | Update information and conclusions |
| 11/12/2022 | V0.3 | YIZHOU FENG/YONGXIANG HUANG | Final fix |

Table of Content

Contents

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: 1520887

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 performing static code analysis using the ION Open Source 4.1.1 dashboard for CID 1520888, this vulnerability is flagged as a high severity vulnerability and we believe the presence of this vulnerability poses a high risk to all code. Also, the presence of the return\_constant flaw and the overrun-buffer-arg.

Observations

*For* **return\_constant error**

The function zco\_clone() appears to be a clone of a Zero-Copy Object (ZCO), which is a data structure used in some software systems to manage memory and optimize data transfer performance. The function call you provided includes four arguments: sdr, work->zco, work->zcoBytesConsumed, and work->bundleLength. It's not clear what these arguments represent or what the function does with them, so it's difficult to say exactly why the function might return the value 18446744073709551615.

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.

For **assignment error**

As mentioned above, the function zco\_clone() appears to be a clone of a Zero-Copy Object (ZCO), so assigning the result of this function to the variable work->rawBundle would create a new ZCO object that is a copy of the original work->zco object. If the function call you provided returns the value 18446744073709551615, then this value would be assigned to work->rawBundle.

It's not clear what the significance of this value is or why the function might return it, but it's possible that it indicates some kind of error or exceptional condition. Without more information about the context in which this code is being used and the intended behavior of the zco\_clone() function, it's difficult to say for sure.

*Overrun-buffer-arg error.*

CID 1520887 is an identifier used by the Common Vulnerabilities and Exposures (CVE) system to track publicly known cybersecurity vulnerabilities. The associated description, "Out-of-bounds access (OVERRUN)," indicates that the vulnerability involves an issue with accessing data outside of the boundaries of an allocated memory area. This type of vulnerability can be exploited by attackers to gain unauthorized access to sensitive data or to crash a system. It's important for organizations to regularly scan for and address known vulnerabilities in their systems to prevent attacks.

For **overrun-buffer-arg error**

The function zco\_destroy() appears to be a function that destroys a Zero-Copy Object (ZCO), which is a data structure used in some software systems to manage memory and optimize data transfer performance. The function call you provided includes two arguments: sdr->dssm and work->rawBundle. The work->rawBundle argument appears to be a ZCO object that was created by calling the zco\_clone() function, as described in your previous question.

The issue you've raised is that the value of the work->rawBundle argument, 18446744073709551615, is very large. This value is the maximum value that can be represented using 64 bits, so it's possible that it was assigned to work->rawBundle as a result of an error or exceptional condition. If this is the case, then calling the zco\_destroy() function with this argument could cause issues, such as an attempt to access memory outside of the allocated range for the ZCO object.

It's also possible that the value of work->rawBundle is not an error, but rather a valid index into some kind of data structure that is being used by the zco\_destroy() function. However, without more information about the context in which this code is being used and the intended behavior of the zco\_destroy() function, it's difficult to say for sure.

Supporting Evidence  图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成



图形用户界面, 文本, 应用程序

描述已自动生成

Conclusions and Recommendations

*In general, if you believe that your organization may be vulnerable to the CID 1520887 (OVERRUN) vulnerability, it's important to take steps to address the issue as soon as possible. This may involve conducting a thorough security review of your systems, applying any relevant patches or updates, and implementing additional security measures to prevent unauthorized access. It may also be a good idea to consult with cybersecurity experts who can provide guidance and assistance.*

One of the most effective ways to prevent out-of-bounds accesses and the potential security vulnerabilities they can introduce is to carefully check that array subscripts are within the legal bounds of the array and that the array is not accessed beyond its valid memory area. This can help to ensure that the program only accesses data that is within the allocated memory area and to prevent attempts to access sensitive data or crash the system. In addition to checking array subscripts, it's also important to use secure coding practices and to regularly review and update your systems 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:

1. 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.
2. 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.
3. Use appropriate methods to prevent buffer overflows. For example, if you are using the strcpy function to copy strings into an array, make sure that the destination array is large enough to hold the entire string.
4. Regularly review your code to ensure that it is free from potential out-of-bounds accesses. This can include conducting code audits, using static analysis tools to identify potential vulnerabilities, and testing your code thoroughly.
5. Update your systems regularly to apply any relevant patches and security updates that address known vulnerabilities.

By following these steps, you can help to ensure that your arrays are not accessed beyond their valid memory areas and to prevent potential security vulnerabilities.

References   
*Please keep an updated references list in APA7; The Deakin referencing guide can be found* [here](https://www.deakin.edu.au/__data/assets/pdf_file/0009/2236752/Deakin-guide-to-APA7.pdf)*.* 

Appendix

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