Coverity Scan Static Analysis Report

Hardhard Enterprises

**T3** 2022

**T3** 2022

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** |
| 20/03/2023 | V0.1 | Callam Besley | Initial Investigation |
|  |  |  |  |
|  |  |  |  |

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 ***Memory Corruptions*** type defect identified in the following CIDs:  
***1520851***

# 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 |
| ASLR | Address Space Randomisation |
|  |  |

# Code Review and Analysis

## Outcomes

The Coverity Report identified multiple memory corruption issues. Analysis of the code revealed that the code intends to process bundles for transmission over a network. The code intends to **canonicalicalize** the primary and payload blocks of a **bundle** in the interplanetary Internet protocol through converting them into a standardised format for storage and transmission.

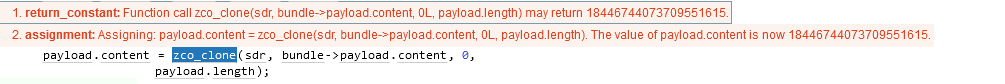
## Observations

Observation of the code revealed that there is an out-of-bounds access in the program **(buffer overflow)**, specifically an overrun error as the program is attempting to read/write data outside the range of the buffer (the amount of memory allocated for temp storage of data). This may cause issues such as node crashes, creation of a cyber exploit entry point and cause adjacent storage to store the overflowed data. It should be noted that programs written in C are more vulnerable to buffer overflow attacks due to a lack of safeguards against overwriting of data.

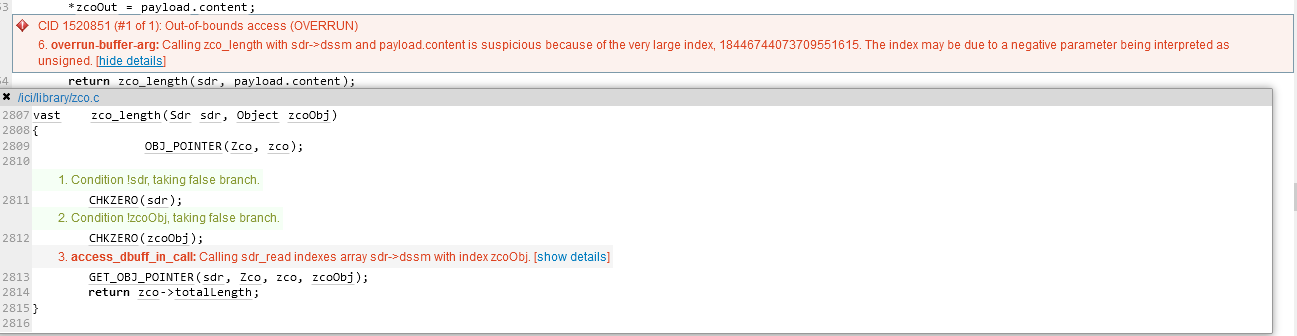
The index of 18446744073709551615 may be a result of a negative parameter being interpreted as unassigned. Its likely that payload.content has not been allocated enough memory to hold data being processed causing **out of bounds access**. The negative parameter being read as unassigned may be due to an issue with **type casting** or a mismatch between actual data types used in the program.

The code is exploitable.

## Supporting Evidence

Based on the error message, payload.content is being accessed using a very large index which then causes the buffer overflow.

The root issue is zco\_clone is returning a very large index value which causes the out of bounds access error further in the code as payload.content is assigned this integer. This is due to a negative parameter being interpreted as an unassigned value as the error message suggests.

This seems to be the main error, the compiler is drawing upon indexes outside of the specified array **(out of bounds access)** the odd index number 18446744073709551615 is being compiled, and thus causes the **buffer overflow.**

# Conclusions and Recommendations

Its clear that a buffer overflow is a great cause for concern as it has exploit potential and hence must be fixed immediately. Attackers can exploit this by intentionally overwriting the memory of an application, changing the execution path of the program allowing them to trigger a response that either damages files or expose sensitive information. For example an attacker can implement extra code into executeable code that gives them access to the ION, or even crash the ION.

**Recommendations:**   
Ensure all buffers are properly initialized and have adequate space to hold the data being processed.

Use ASLR, this runtime protection shifts the address space locations of data regions. Since buffer overflow attacks tend to require knowledge of the location of executable code, this makes it significally harder for them to buffer overflow their own code into overwriting executeable code.

Flag areas of memory as non-executable which prevents an attacker from running code in an executable region.

The Coverity static analysis scan program itself identifies red flags for potential buffer overflows, which are then triaged and fixed individually.

Consider use of dynamic memory allocation, such as malloc() and dalloc() which allocate memory based on what you specify, in this case it is not needed.

In terms of the error itself, consider changing the payload.length variable to an unassigned integer so that the compiler will always interpret it as an unsigned value and prevent any negative values being passed to the zco\_clone function, alternatively, we could include the check:

If (payload.length < 0) {

putErrmsg(“Invalid payload length”, NULL);

return -1;

}

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).  
Rouse, M. (2011, October 5). *Canonicalization.* Techopedia. <https://www.techopedia.com/definition/15646/canonicalization>

Christensson, P. (2006) *Buffer*. TechTerms.com <https://techterms.com/definition/buffer>

Imperva. (n.d.) *Buffer overflow attack.* <https://www.imperva.com/learn/application-security/buffer-overflow/>

Priya, B (2021, March 9). *What is out of bounds index in an array.* TutorialsPoint.https://www.tutorialspoint.com/what-is-out-of-bounds-index-in-an-array-c-language

BYJU’S EXAM PREP (n.d.) *Type Casting in C.* https://byjus.com/gate/type-casting-in-c/

Appendix

# **Canonicalize –** The process of converting data that involves more than one representation into a standard approved format. This ensures that the data conforms to canonical rules.

**Bundle –** A package/collection of related files that are grouped together and can be easily distributed, installed and managed as a single unit.

**Buffer –** Holds data, storing it for a short amount of time before use.

**Buffer overflow –** When the volume of data exceeds the storage capacity of the buffer which causes the program that writes the data to override adjacent memory

**Out of bounds access –** When an array index is included and used when it is out of bounds, causing the compiler to run it, the output is typically incorrect.

**Type Casting –** When a variable belonging to one data type is changed to another data type, in type casting, the compilier automatically changes one data type to another based on what we require the program to do.