Coverity Scan Static Analysis Report

Hardhard Enterprises

**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** |
| 2/04/2023 | V1 | Jesse Ludeman | Initial document and investigation |
| 5/04/2023 | V1.2 | Jesse Ludeman | Continue investigation |
| 6/04/2023 | V1.3 | Jesse Ludeman | Finalize 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: 1520765

# 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

## Introduction

CID 1520765 is an Out-of-bounds type defect in the [/bpv7/library/ext/bpsec/bcb.c] file which is subsequently located in the bcbUpdatePayloadFromSdr function. This type of weakness means that the program is attempting to perform an operation on a memory buffer, but it can still read from or write to a memory location that is outside of the intended boundary of the buffer.

There is also an additional error that has been flagged on line 423 which we investigate first.

## Observations

The function bcbUpdatePayLoadSdr(), as the name suggests is designed to update the Block Confidentiality Block (BCB) payload from the Simple Data Recorder (SDR).

The first issue that we notice is the return\_contant error from the zco\_create() function, as noted in figure 1.

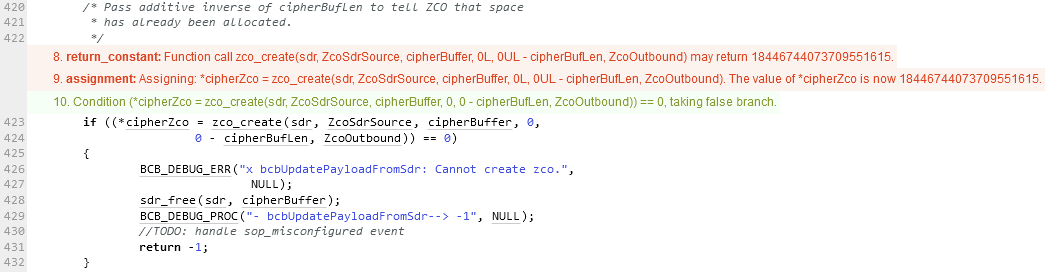


Figure 1 - The defect is flagged on line 423

The initial error occurs on line 423 as noted in figure 1 in the if statement, where a call to the zco\_create() function is made. Whilst this function has no commentary added and limited documentation is available, it appears that this function creates a new Zero-Copy Object (ZCO) in the context of the DTN protocol. A ZCO refers to a technique in programming where data is transferred between differing parts of a computer system without being copied from one memory location to another. Copying data from one memory location to another can be a time consuming and resource intensive process.

In the DTN protocol, a ZCO is a data structure that provides a way to efficiently handle large amounts of data in a low-bandwidth network environment. When a large data chunk needs to be transmitted on the DTN network, it may be divided up into smaller chunks called “bundles”. Each bundle contains a ZCO that points to the corresponding chunk of data.

The second issue that we notice is the Out-of-bounds access (OVERRUN) error as noted in figure 2 on line 488. A call to zco\_destroy() is made, which passes in the sdr and \*cipherZco variables. This function will destroy the object residing at location \*cipherZco immediately if it exists. Otherwise, it flags the object for destruction as soon as the reference to it is removed.



Figure 2 - Out-of-bounds access (OVERRUN) error

Supporting Evidence  
Observing the return\_constanterror on line 423, we notice that the \*cipherZco pointer is assigned the value of the zco\_create() function in the if statement on line 423. The zco\_create() function *may* return 18446744073709551615. Due to this, the value of \*cipherZco will now be assigned 18446744073709551615. This conditional check is validating whether \*cipherZco will evaluate to zero (0). Given that this situation will not occur in this particular scenario, the body of the if statement never executes. This normally wouldn’t be a problem, and could be by design. However, adding an input validation check to zco\_create() may resolve this defect.

For the Out-of-bounds access issue we’re investigating, we’re advised that this operation is suspicious because of the very large index which is 18446744073709551615. We know that from our previous operation that the variable \*cipherZco is assigned this large number, therefore; it’s not the call to sdr that is the problem.

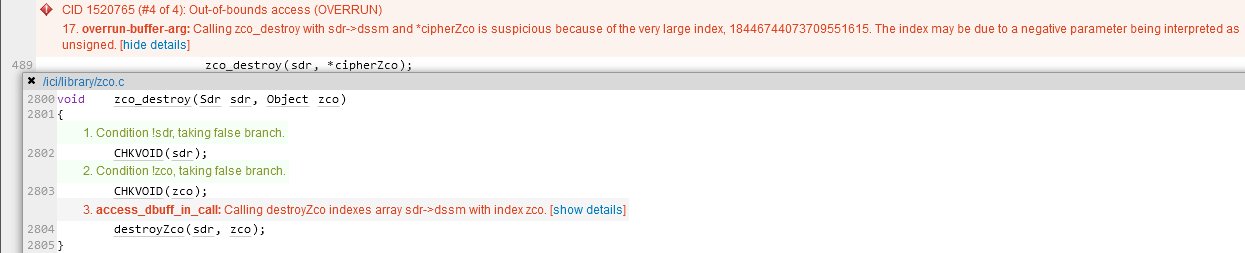


Figure 3 - zco\_destroy() function

Inspecting the zco\_destroy() function, it briefly advises us that the call to destroyZco() indexes the sdr array with index zco. Based on our previous investigation, this means that the return value from zco\_create() could be 18446744073709551615. Given that this value was assigned to the \*cipherZco variable, which is then passed to the zco\_destroy() function; this means that this value is being passed as an argument to this function. As a result, Coverity has flagged this as a defect.

# Conclusions and Recommendations

Both of these errors in this defect are not vulnerabilities. However, these errors can be fixed with mitigating code. For example, input validation could be added to the zco\_destroy() function such that if the zco argument that’s passed to it is too large, then handle the scenario accordingly.

|  |
| --- |
| #define ZCO\_SIZE 1024  void zco\_destroy(Sdr sdr, Object zco)  {      // Handle the input      if (sizeof(zco) >= ZCO\_SIZE)      {          printf("ZCO object is too large");          return 0;      }      CHKVOID(sdr);      CHKVOID(zco);      destroyZco(sdr, zco);  } |

Figure 4 - Potential fix for zco\_destroy() function

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.