**T1** 2023

Coverity Scan Static Analysis Report

Hardhard Enterprises

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** |
| 18 Apr 2023 | V0.1 | Connie Cox | Initial draft. |
|  |  |  |  |
|  |  |  |  |

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*** category with an ***Out of bounds access*** type issue identified in the following CIDs:

## 1520781 [Out-of-bounds access](https://scan7.scan.coverity.com/doc/en/cov_checker_ref.html#static_checker_OVERRUN)

## 1520787 [Out-of-bounds access](https://scan7.scan.coverity.com/doc/en/cov_checker_ref.html#static_checker_OVERRUN)

## 1520847 [Out-of-bounds access](https://scan7.scan.coverity.com/doc/en/cov_checker_ref.html#static_checker_OVERRUN)

# Acronyms and Abbreviations

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| DTN | Delay/Disruption Tolerant Network |
| ELT | Element |
| ION | Interplanetary Overlay Network |
| PSM | Personal Space Memory |

# Code Review and Analysis

## Overview

The Coverity report identified several Out-of-bounds access issues with the bpsec\_policy\_rule.c within the Bundle Protocol v7 library extensions for bundle protocol security [/bpv7/library/ext/bpsec] directory. The following analysis outlines the issues identified in the following functions:c

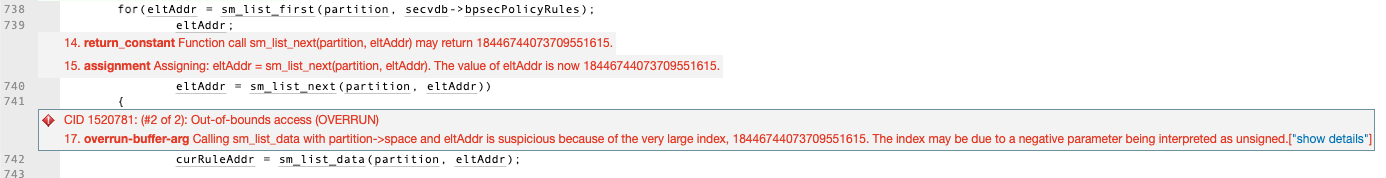
* bslpol\_rule\_remove – removes a rule from the BPSec Policy Engine. When the rule is removed, it must not be referenced by the calling function and it will alter the index of every rule before it in the list.
* bslpol\_scparm\_find – finds the bpsec policy rule that best matches the blocked identified by the block number (tgNum), the security context ID (SCID) for the security block and the role of the security verifier or acceptor (BPA)
* bslpol\_cb\_ruleradix\_remove – removes a rule from a radix tree node where the key for a particular radix tree is the EID being index by that tree. The value of the EID is a set of all the rules which match that EID.

## Observations

The bpsec\_policy\_rule file implements processing specific to BPSec policy rules which are used by the policy engine to associate security actions with security events. The functions underpinning the CIDs are bslpol\_rule\_remove(), bslpol\_scparm\_find() and bslpol\_cb\_ruleradix\_remove().

**FUNCTION:** bslpol\_rule\_remove()

The bslpol\_rule\_remove() function returns a success or error value to indicate the removal of the rule from the list based on the shared memory partition and the rule address of the rule provided



In the above code (line 742), the element address denoted by eltAddr is of data type PsmAddress and is expected to be returned on assignment to sm\_list\_next(). On closer inspection, if the partition that is passed into the bslpol\_rule\_remove() is null, and/or the address pointer from bpsecPolicyRules (line 738) is null, then when these parameters are used to retrieve the next rule address in the list, a large vaule is assigned to eltAddress. This can be see in Coverity (line 740 in screenshot below) where the root cause of the issue is likely to be the function querying an invalid list thus returning a large number.



**FUNCTION:** bslpol\_scparm\_find()

The bslpol\_scparm\_find() function returns the PSM Address of a rule based on:

* the partition address
* the PSM parameters and;
* the type of rule.

In this function, the pointer to the rule’s memory address is retrieved from the linked list by using the sm\_list\_first() and assigned to a PSM Address variable called elt (line 961). As with other similar assignments across the bundle protocol, there does not appear to be any validation of this address on assignment, however, the variable is used in line 963.



Coverity at line 963 indicates an out-of-bounds access (overrun) issue because of this.



**FUNCTION: bslpol\_cb\_ruleradix\_remove()**

In this function, a rule is removed from a radix tree that is associated to a specific key-value pair, where the key is the EID being indexed by the radix tree and the value is the set of rules that match the key. Accordingly, when removing a rule from an indexing radix tree, the rule to be removed is removed from the set of rules at the node. In order to do this, the key inputs are, the shared memory partition (PSM address) of the node, the address of the set of rules for the node and the address of the rule being removed must be known when passed to this function. In simple terms, this function removes an address from the list of rules at the node that matches the input parameters of the function. However, the address itself is not deleted because the radix tree only holds the addresses to the rules that are elsewhere in memory.

As with all the other functions that use linked lists, Coverity raises an issue because of the assignment index to eltAddr. This issue can occur when at the time of running the scan, there were no addresses being stored in the linked list making the linked list invalid or that the PSM address of the node or rules at input were invalid.

## Supporting Evidence

### What is an out of bounds access (overrun) issue?

An out-of-bounds access error occurs when the linked list accesses memory and/or stores data beyond its original memory allocation.  In this context, the index used in the linked list storing PsmAddresses for span elements could potentially be larger than the max index that was allocated to the original linked list.  This has the potential to:

* Cause adjacent storage to store overflowed data
* Crash the node(s)
* Create an entry point for a cyber exploit

### What is a radix tree?

According to (Morrison, 1968), a PATRICIA (Practical Algorithm to Retrieve Information Coded in Alphanumeric) tree is also known as radix tree. Each node in a radix tree represents a character in the key or a partial key where the edges that connect the nodes are are labelled with the characters. The leaf nodes store only the keys. The key difference between a radix tree and a trie is that a radix treen can represent complete keys making them more space efficient.

# Conclusions and Recommendations

When elements are used with linked lists, there needs to be a check to ensure that the element exists before it is added to the linked list. This would ensure that the linked list is valid and that there isn’t a buffer overrun on assignment to PsmAddresses.

There also needs to be validation when items are retrieved or removed from the list and assigned to a PSM Address variable. This ensures that any pointer returned for a PSM Address is valid. This also ensures that the linked list used is valid and there isn’t a buffer overrun on assignment to a PSM Address.

The key issue of these three CIDs is the lack of validation of input and/or retrieval of values prior to assignment to variables for further use.

# Bibliography

Morrison, D. (1968). PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Journal of the ACM, 15*(4), 514-534.

Appendix

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