**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** |
|  | V0.1 | Jaspriya |  |
|  |  |  |  |
|  |  |  |  |

Contents

Table of Content

[Introduction 3](#_bookmark0)

[Objective 3](#_bookmark1)

[Scope 3](#_bookmark2)

[Acronyms and Abbreviations 3](#_bookmark3)

[Code Review and Analysis 4](#_bookmark4)

[Outcomes 4](#_bookmark5)

[Observations 4](#_bookmark6)

[Supporting Evidence 4](#_bookmark7)

[Conclusions and Recommendations 4](#_bookmark8)

[References 5](#_bookmark9)

[Appendix 6](#_bookmark10)

# 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 ***‘Untrusted bound as a loop’*** type defect identified in the following CIDs:

***1520690***

***1520691***

# 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 |
| CWE | Common Weakness Enumeration |
| EIDLEN | Endpoint identifier length |

# Code Review and Analysis

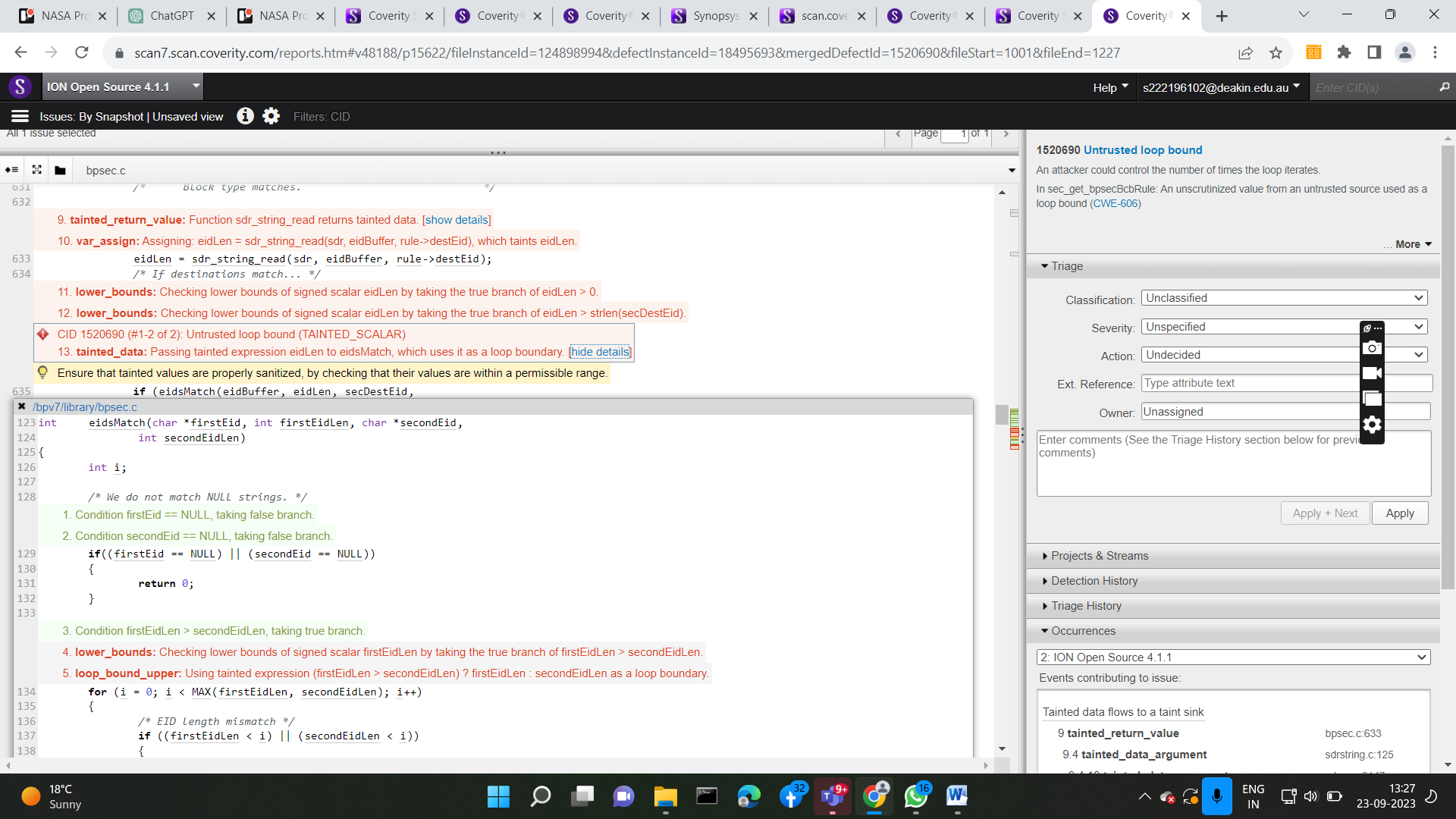
Overview

Coverity has flagged CID 1520690 and CID 1520691 as a code issue in the **serializeDataPDU** function. This issue is categorized as a potential security concern due to the presence of an unvalidated input value originating from an untrusted source. Specifically, the **sdr\_string\_read** function is returning tainted data, which is then assigned to the variable **eidLen**. The tainted **eidLen** is subsequently used as a loop boundary in the **eidsMatch** function.

## Observations

CID 1520690:

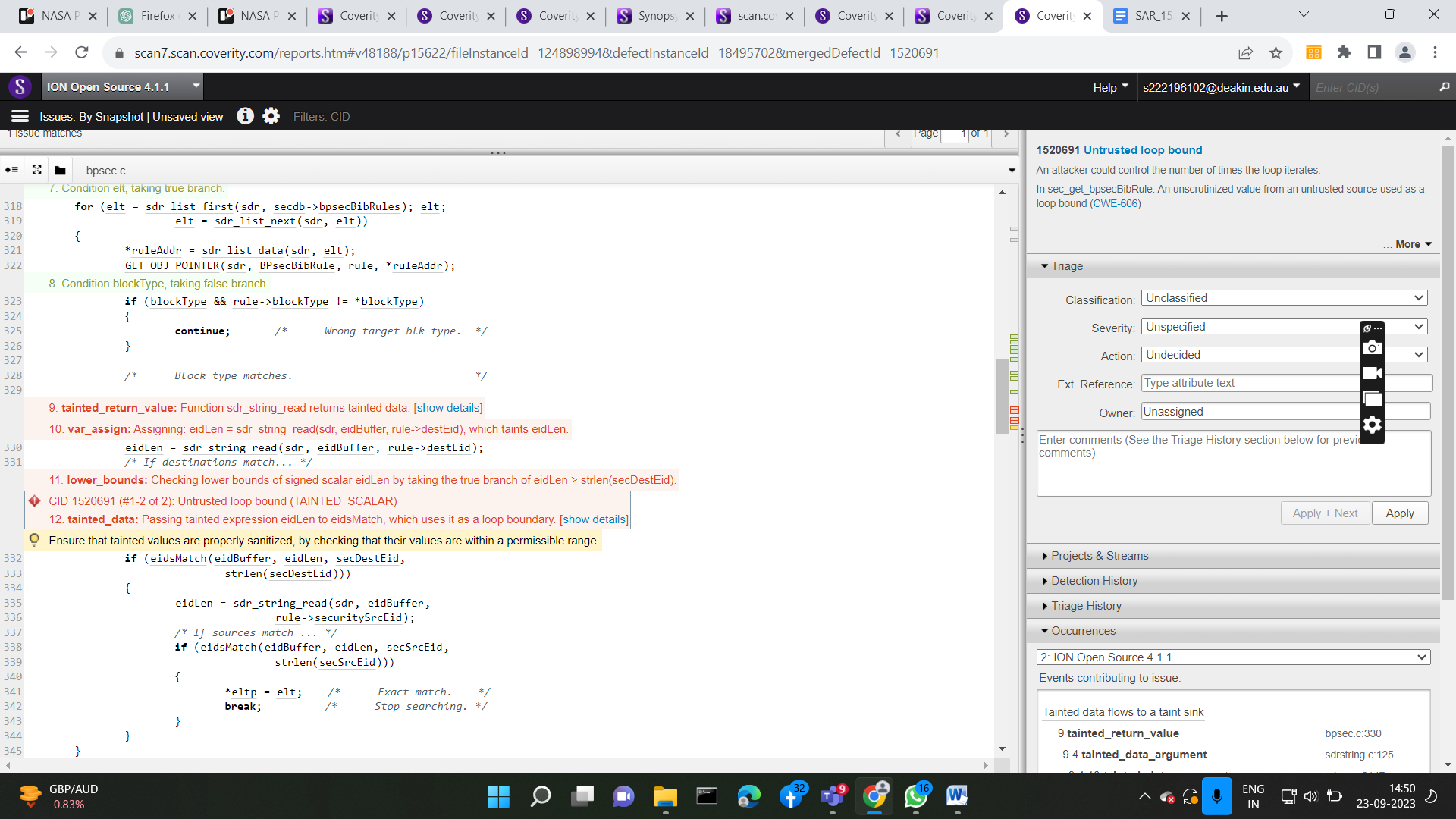
* **Tainted Data (CID 1520690 - tainted\_return\_value):** The sdr\_string\_read function returns tainted data, as indicated by CID 1520690. This implies that the data returned by this function may originate from an untrusted or potentially malicious source.
* **Tainted Variable Assignment (CID 1520690 - var\_assign):** Line 633 assigns the result of sdr\_string\_read to the variable eidLen, tainting it in the process. The tainting of eidLen suggests that it may contain untrusted or potentially unsafe data.
* **Lower Bounds Checks (CID 1520690 - lower\_bounds**): The code contains lower bounds checks for the eidLen variable. Specifically, it checks if eidLen > 0 and eidLen > strlen(secDestEid). These checks are intended to validate the value of eidLen and ensure that it meets certain criteria.
* **Untrusted Loop Bound (CID 1520690 - tainted\_data):** A critical concern is raised regarding the use of the tainted eidLen variable as a loop boundary in the eidsMatch function. This indicates that eidLen may control the execution of a loop, and the use of tainted data in this context can lead to security vulnerabilities.



**CID 1520690**

CID 1520691:

* **Tainted Data (CID 1520691 - tainted\_return\_value):** The **sdr\_string\_read** function is identified as returning tainted data (CID 1520691). This implies that data returned by this function may originate from an untrusted or potentially malicious source, raising security concerns.
* **Tainted Variable Assignment (CID 1520691 - var\_assign):** At line 330, the result of **sdr\_string\_read** is assigned to the variable **eidLen**, tainting it in the process. This suggests that **eidLen** may contain untrusted or unsafe data, posing a potential security risk.
* **Lower Bounds Checks (CID 1520691 - lower\_bounds):** The code contains lower bounds checks for the **eidLen** variable. Specifically, it checks if **eidLen > strlen(secDestEid)**. These checks aim to validate **eidLen** and ensure it meets specific criteria.
* **Untrusted Loop Bound (CID 1520691 - tainted\_data):** A significant concern is raised regarding the use of the tainted **eidLen** variable as a loop boundary in the **eidsMatch** function. This indicates that **eidLen** may control the execution of a loop, and the use of tainted data in this context can lead to security vulnerabilities.



**CID 1520691**

## Supporting Evidence

# Conclusions and Recommendations

Conclusion: The identified security vulnerability presents a potential avenue for attackers to manipulate the eidLen variable with arbitrary inputs, posing a significant security risk. To mitigate this issue, it is imperative to implement stringent input validation and sanitization mechanisms. Such measures will enhance the system's security and thwart potential exploits effectively.

Recommendation: Implement robust input validation checks for the eidLen variable and associated inputs to ensure they conform to safe and expected ranges. Additionally, consider a security review of the sdr\_string\_read function to ensure it properly sanitizes inputs and returns untainted data. This proactive approach will fortify the system against security threats.

# Real Life Scenario

In real life, the firmware of a medical device has a problem called "Untrusted Loop Bound" and "Tainted Return Value" that affects a very important feature called "calculateDosage." Based on information about the patient, this function figures out how much of a drug to give. Line 330 uses a function to get the patient's weight, but it is marked as giving possibly bad data. Line 331 checks to see if the patient's weight is correct, and line 11 makes sure it isn't negative. But the problem happens when this possibly bad number is used in the 'calculateDosage' function to figure out how much medicine to give. If not treated correctly, this could lead to patients getting the wrong doses, which could put their health and safety at risk. Also, the "Untrusted Loop Bound" problem could lead to holes in the logic of the loop, which could cause the dosage calculation to act in an unexpected way and put the patient's health at risk. These problems show how important it is to fix security and correctness problems in software for medical devices so that mistakes that could kill people don't happen.

# Latest three solutions:

Solutions for Untrusted Loop Bound Vulnerability:

[1] **Check your bounds.** Untrusted loop bounds vulnerabilities can be found and avoided with this tool.

[2] **Employ secure coding practices.** Writing code that exploits these vulnerabilities is difficult or impossible in some programming languages, such as Swift and Rust.

[3] **Employ secure coding practices.** You can develop secure code using this set of instructions and resources, which also includes advice on how to prevent untrusted loop boundaries vulnerabilities.

Untrusted Bound as a Loop Vulnerability Statistics:

* Most common static analysis finding (accounting for over 10% of all findings)
* Critical security vulnerability that can be exploited to cause denial-of-service attacks, buffer overflows, and arbitrary code execution
* Difficult to detect and fix

Mitigation Techniques

* Validate all input data
* Use a bounds checker
* Use a safe programming language
* Use a secure coding framework

References

[1] Wikipedia contributors, "Bounds checker," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/BoundsChecker, accessed September 23, 2023.

[2] The Rust Programming Language Foundation, "The Rust Programming Language," https://www.rust-lang.org/, accessed September 23, 2023.

[3] Apple Inc., "Swift," https://developer.apple.com/swift/, accessed September 23, 2023.

[4] Carnegie Mellon University Software Engineering Institute, "CERT C Coding Standard," https://wiki.sei.cmu.edu/confluence/display/c/, accessed September 23, 2023.

# Appendix

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