**T2** 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 | Kanad Dombhare |  |
|  |  |  |  |
|  |  |  |  |

Table of Content

Contents

Introduction 3

Objective 3

Scope 3

Acronyms and Abbreviations 3

Code Review and Analysis 4

Overview 4

Supporting Evidence 4

Conclusions and Recommendations 6

# 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 Divisor*** type defect identified in the following CIDs:  
1520886 & CID 1520731

# 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

## Overview

Coverity has discovered an medium impact vulnerability ie Untrusted divisor, an "untrusted divisor" refers to a variable or value used as the denominator in a division operation, but its reliability or safety cannot be guaranteed. This situation can lead to division-by-zero errors or unexpected behavior in the program. Handling untrusted divisors is crucial to prevent crashes or security vulnerabilities.

Observations

According to CWE (Common weakness Enumeration )-369

This weakness typically occurs when an unexpected value is provided to the product, or if an error occurs that is not properly detected. It frequently occurs in calculations involving physical dimensions such as size, length, width, and height.

The code shows the use of tainted expression “keyLength” . "Tainted values" or "tainted data" refer to untrustworthy or potentially dangerous input or data. This word is often used in security and vulnerability assessments to discover prospective data sources that could lead to security breaches or unexpected behavior in a software. When data is considered tainted, it means that the data comes from an external or untrusted source, such as user inputs, data read from a file, data received over a network, or any data that is not under the direct control of the program. A hacker may edit or create contaminated data to take advantage of software flaws.Tainted data becomes a security concern when it is used in sensitive operations or when it is used without proper validation and sanitization. If tainted data is not properly handled, it may lead to issues.

# Supporting Evidence

## Screenshot 2023-08-06 at 9.51.40 PM.png

*Fig 1.1 (CID1520886)*

In the following code, we can see both the CID’s show the same issue, we can see there is a tainted expression which relies on tainted data , as as mentioned above it comes from untrusted sources and can be potentially manipulated or crafted by attacker to exploit vulnerabilities . If the tainted expression is used without proper validation it can lead to security vulnerabilities

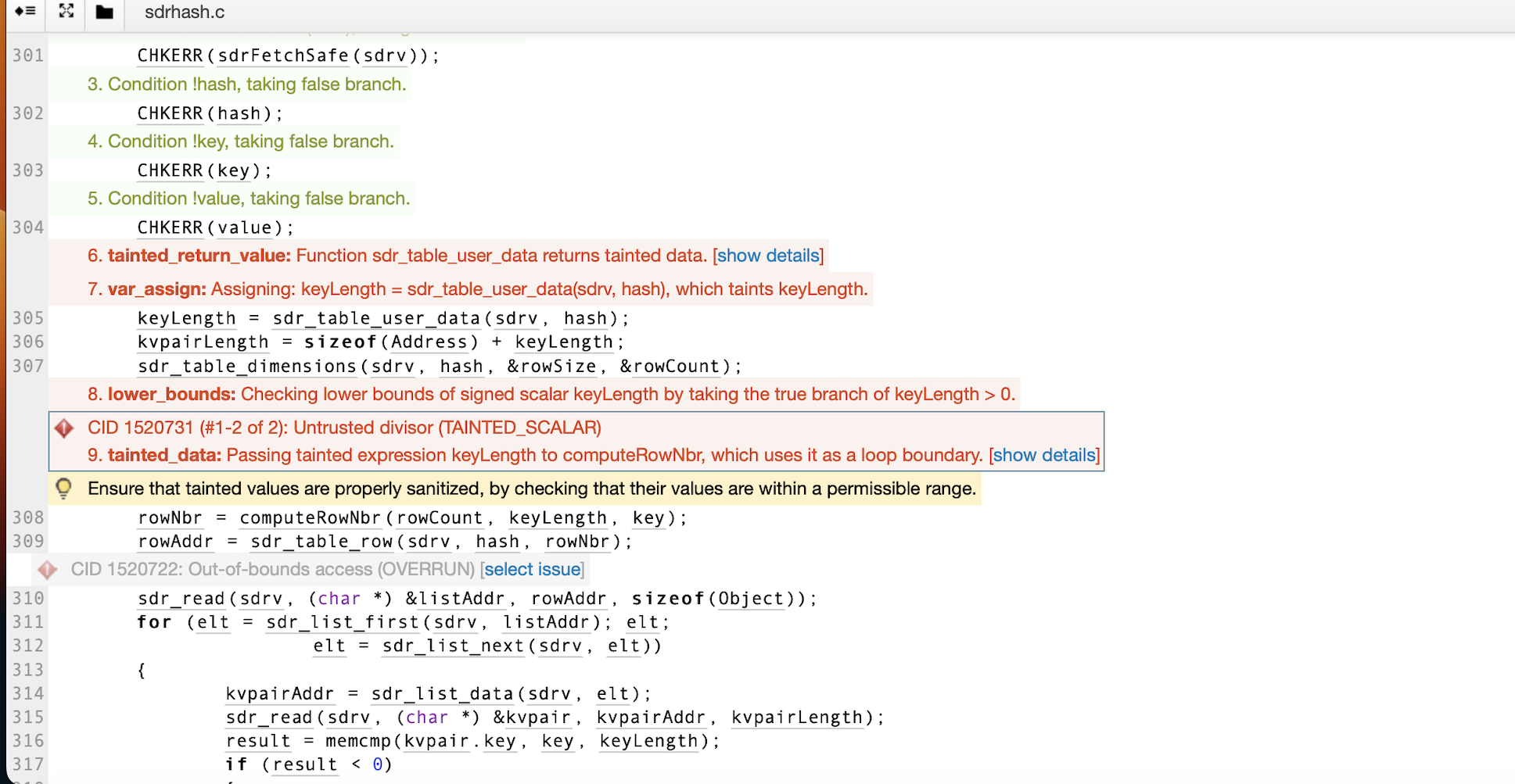
This means that the loop will iterate **keyLength** times, potentially leading to issues if the value of **keyLength** is not correctly validated and sanitized.

The potential security implication of using tainted data are

**Loop bound errors** : A buffer overflow or out-of-bounds access could result from an attacker manipulating the tainted keyLength value to be excessively large or negative. Alternatively, the loop may not iterate at all, which could cause unexpected program behavior.

**Resource Exhaustion**: If an attacker provides a very large value for **keyLength**, it could cause the loop to consume excessive resources (e.g., CPU time and memory) and potentially crash the program or cause a denial-of-service (DoS) condition.

Also it shows stained scalar which refers to single digit values marked as tainted or untrusted. It specifically refers to individual scalar data values such as integers, floating-point numbers, characters or strings.

**

*(FIG 1.2 CID 1520731)*

# Conclusions and Recommendations

Handling untrusted divisors is crucial to prevent crashes or security vulnerabilities. Developers should validate and sanitize input values, especially those used in division operations, to ensure they are within permissible ranges and non-zero before performing the division. Proper input validation and error handling are essential to maintain the stability and security of the code. Ignoring untrusted divisors can lead to potential bugs or exploits, making it vital to treat them with caution and implement appropriate safeguards.

Developers should validate and sanitize input values, especially those used in division operations, to ensure they are within permissible ranges and non-zero before performing the division. Proper input validation and error handling are essential to maintain the stability and security of the code. Ignoring untrusted divisors can lead to potential bugs or exploits, making it vital to treat them with caution and implement appropriate safeguards.

*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.*