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 formatted 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** |
| 7/12/2022 | 1.0 | Jesse Ludeman | Initial document |
| 8/12/2022 | 1.1 | Jesse Ludeman | Identify and build mitigation |
| 9/12/2022 | 1.2 | Jesse Ludeman | Investigate mitigation |
| 10/12/2022 | 1.3 | Jesse Ludeman | Identify fix and finalize SAR report |

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 4](#_Toc119848733)

[Appendix 4](#_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 ***Assignment of overlapping memory*** type defect identified in the following CIDs: 1520766

# 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

## Outcomes

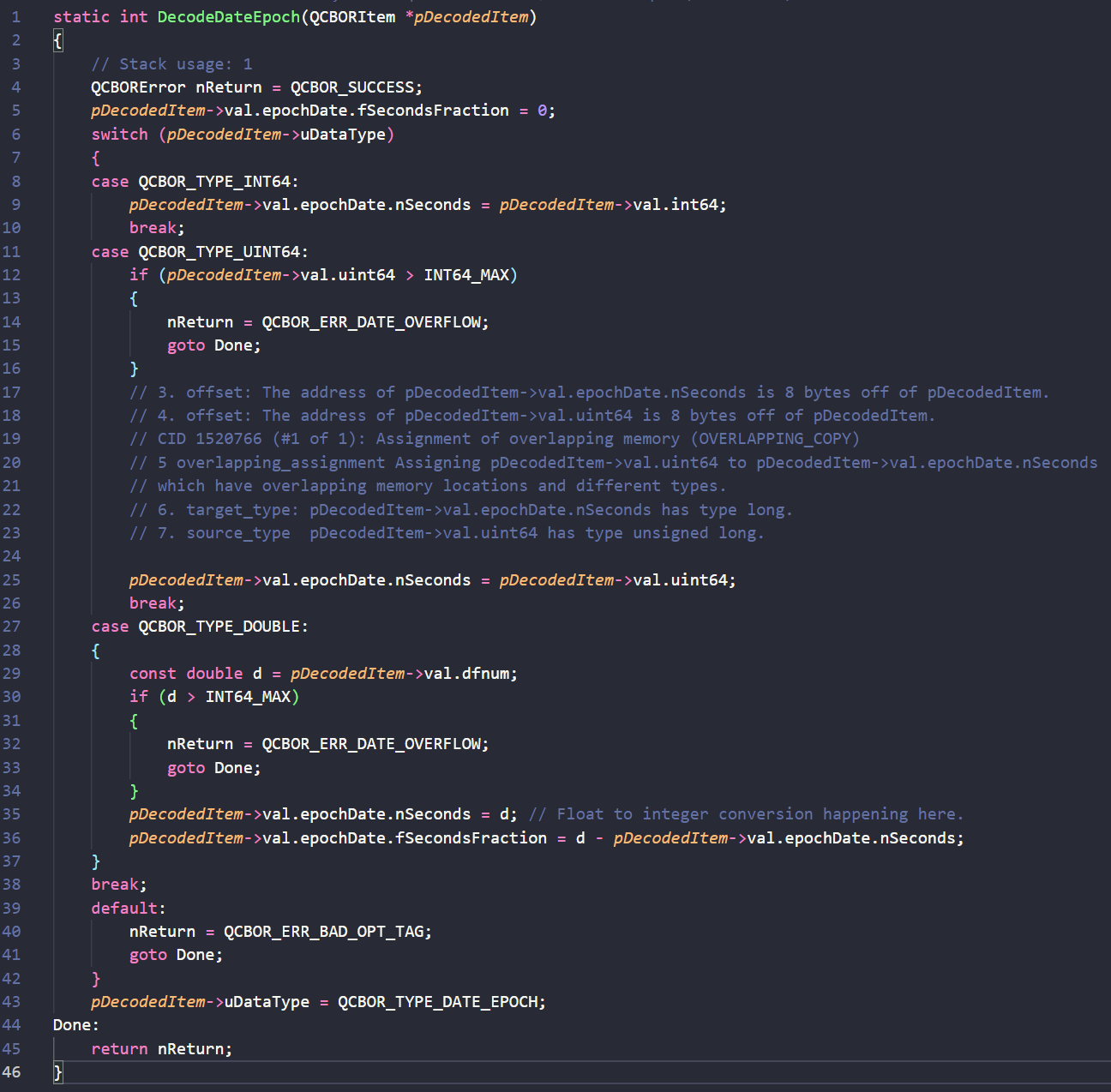
When performing static code analysis using the ION Open Source 4.1.1 dashboard for CID 1520766, there is a high impact problem that involves the assignment of overlapping memory in the decodeDateEpoch static function of the qcbor\_decode.c file.

## Observations

The OVERLAPPING\_COPY error indicates that data is being copied to an overlapping location in memory. This can occur during assignment or when the memcpy function is being called. Assignment will result in an “undefined manner”, unless the overlap is considered to be exact, and the data types are compatible. When memcpy is used, it’s considered to be “always undefined”.

The OVERLAPPING\_COPY error, and memory overlapping issues in general can result in the program reading an incorrect or unexpected value in a given data type.

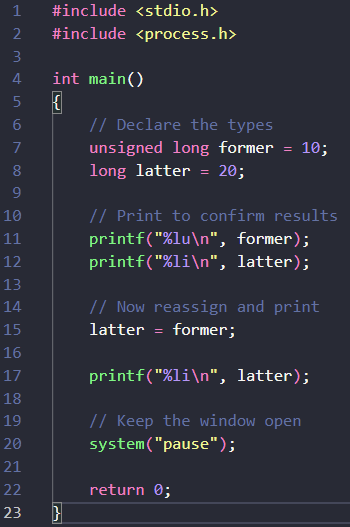
## Supporting Evidence

  
Figure 1 – Line 25 indicates the offending code

Note that in figure 1 on line 1 pDecodedItem is being passed in via reference from the QCBORItem struct. The offending code resides on line 25, where pDecodedItem->val.uint64 is being assigned to pDecodedItem->val.epochDate.nSeconds. The former is of the data type unsigned long, and the latter of type long.

Furthermore, this error has also produced the overlapping\_assignment error. This indicates that an assignment has occurred with potentially undefined semantics. This can cause the program to behave in an unpredictable manner.

For example, unless the overlap is exact and the data types are compatible, then this issue is likely to persist in the C compiler. Figure 2 demonstrates a legitimate use case where both types can be reassigned.

  
Figure 2 – Assigning an unsigned long into a long data type

After investigating the QCBORItem struct, we identify and confirm that that both the uint64 and nSeconds are differing data types. The union data type is also being used, which allows the program to define multiple differing data types that are stored in the same memory location. This means that both data structures occupy the same space in memory. In fact, both data types are part of the same union type.

Text

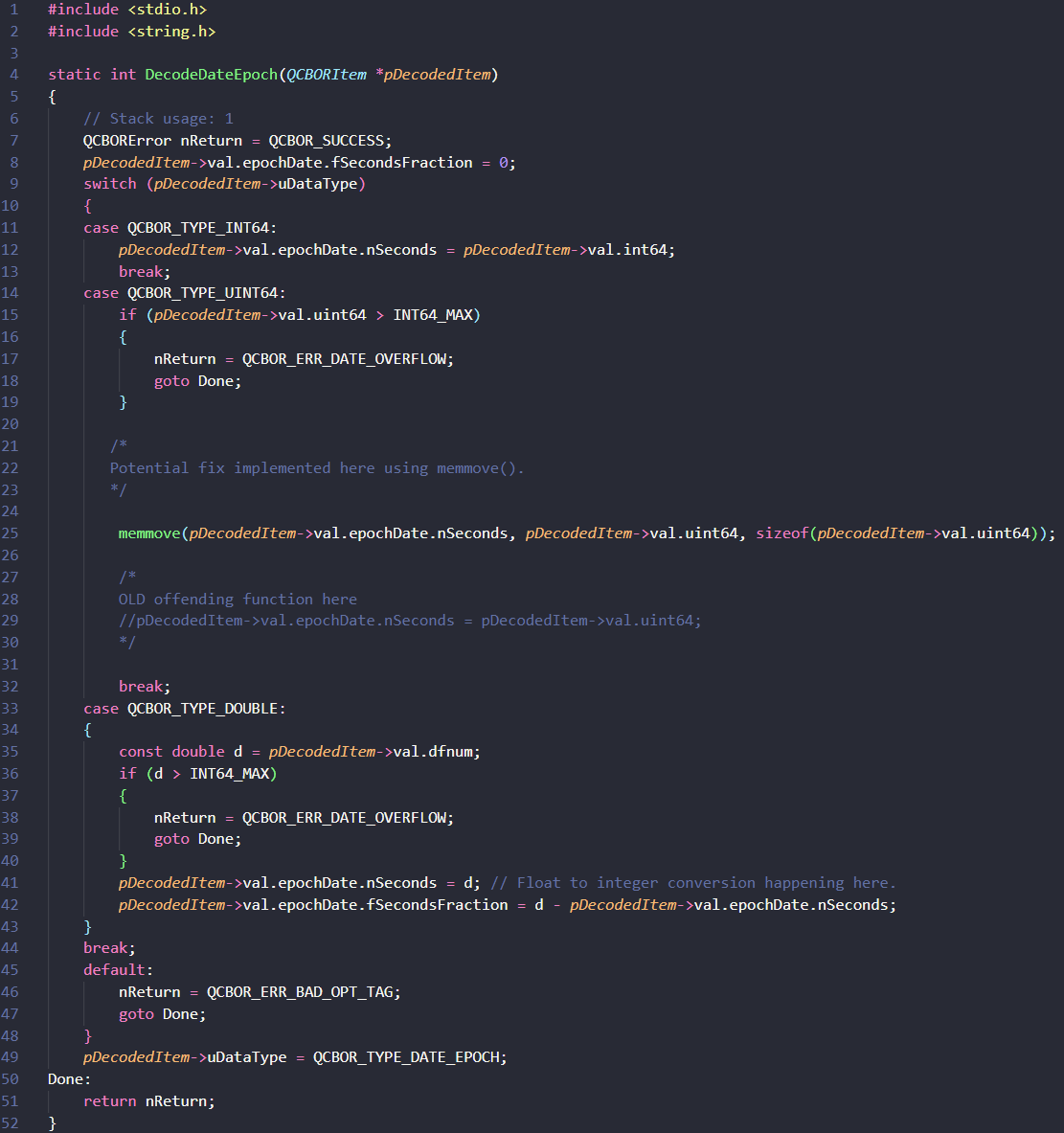
Description automatically generatedFigure 3 – Inspecting the contents of the QCBORItem struct

# Conclusions and Recommendations

Hardhat Enterprises suggests not using the union data type inside the QCBORItem struct. Based on the current program’s design and given that we know that uint64 will be assigned to nSeconds, we suggest not using the union data type in this struct if possible. It is very easy for a programmer to create an overlap using this type.

One solution would be to use the memmove() function instead of directly reassigning uint64 into nSeconds. This is a function in C and C++ that copies a block of memory from one location to another. This effectively allows the source and destination bytes to overlap.

Hardhat Enterprises are satisfied that there is no vulnerability discovered in this report.

  
Figure 4 – Fix implemented using memmove() function

References

Appendix

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