C Secure Coding Standards Performance: CMU SEI CERT vs MISRA

Juan F. García, Miguel V. Carriegos

RIASC, Universidad de León
León, Spain
{juan-felipe, miguel.carriegos}@unileon.es

Jesús Balsa, Fernando Sánchez, Mario Fernández, Alejandro Fernández, Cristian Cadenas, Javier Rodríguez, Vladislav Lebedev
Universidad de León
León, Spain
{jbalsc00, fsanco00, mfernr19, afernc27, ccades00, jrodrv01, vlebed00}@estudiantes.unileon.es

Abstract- **We present a prospective study for performance comparison between programs written in C language and the same programs reviewed and modified to be compliant with CMU SEI CERT C Secure Coding Standard and with MISRA C, the most relevant Secure Coding Standards in existence nowadays. Our initial results show that, as expected, any of the Secure Coding Standards is susceptible to have a negative impact on performance, increasing program running time. We have also found that MISRA C Standard may be less likely to affect code performance than SEI CERT C Standard is, and that it may produce a more optimal code than SEI CERT Standard does; however, further research is needed for proper confirmation of these results.**

***Index Terms*- Secure Coding Standard, CMU CERT, SEI CERT, MISRA, performance**

Tipo de contribución: *Investigación en desarrollo*

I. Introduction

In this research we present a performance review and comparison of the two most relevant C language Secure Coding Standards: Carnegie Mellon University Software Engineering Institute CERT C Secure Coding Standard (CMU SEI CERT C, SC-C from now on) and Motor Industry Software Reliability Association C (MISRA C, M-C).

We have chosen C language since, even nowadays, it is the most relevant high level programming language for the Engineering Community: according to IEEE, C and C++ languages occupy the 1 and 4 rank in the Top Ten Programming Languages for 2016 [1]. They are also the only high level programming languages useful for low level and embedded programming. If web programming is not considered, their edge would even be greater.

On the one hand, C/C++ power comes especially from their low level functions and dynamic memory control, which makes both capable of unmatched performance results among high level languages. On the other hand, both are prone to errors and security vulnerabilities for this very same reasons.

Secure Coding Standards allow us to minimize these errors and security flaws at the early stages of the software development process [2] [3].

However, the secure software we obtain should come with a performance penalty: since the code we generate while following these standard is usually more complex (or at least lengthier than the non-secure one, given the increased number of security checks to include) it is likely to be slower.

SC-C and M-C rival with each other for being the *de facto* C Secure Coding Standard. While both of them try to reduce similar security errors and flaws to prevent vulnerabilities, they differ in their rules and procedures.

In this research we try to find and dynamically measure the performance penalty which should be inherent to make the C code standard-compliant, concluding, if possible, which of them allows for a faster code.

The rest of the paper is organized as follows: section II briefly presents both Secure Coding Standards; section III explains the methodology followed to measure performance; section IV contains a summary of our initial results and section V presents our conclusions and envisions future work.

II. (Secure) Coding Standards

A Secure Coding Standard is a set of guidelines for developing securing code, guarding it against the accidental introduction of bugs and vulnerabilities. Since it usually cover the whole coding spectrum (types, operators, memory usage, preprocessor, error handling, etc.), thus contributing to an overall increase of the code quality, they are sometimes referred to as just Coding Standards.

These standards can be language agnostic or specific, being the Secure Coding Standards for C (and C++) language some of the most developed, and accepted. Among C (Secure) Coding Standard, SC-C and M-C are the most relevant.

SC-C standard consists of rules and recommendations, collectively referred to as guidelines, for C (secure) development [4]. Rules provide normative requirements for C code, whereas recommendations are meant to provide guidance that, when followed, should improve the safety, reliability, and security of software systems.

CMU is the world's leading trusted authority dedicated to security and resilience of computer systems, and an asset in the field of cybersecurity [5]

M-C encompasses guidelines for C language which aim to facilitate code safety, security, portability and reliability [6]. Although initially aimed for embedded systems, it has evolved and is nowadays widely accepted in automotive, aerospace, telecom, medical devices, defense, among others.

III. Methodology

First, we choose a set of problems to be solved in C

language. These problems are solved and validated without considering any Secure Coding Standard, measuring their running time (see III.A for details),.

Then, the original and already validated C programs are reviewed and corrected to make them compliant with each Standard, SC-C and M-C (one at a time). The resulting programs are validated and their running time is measured once again (see III.B for details).

Finally, we perform a statistical study to conclude if there exists any different between the performance of the original and the two standard-compliant versions, and another study to check for differences between both standards (see III.C).

A. *(Non-standard compliant) code performance*

The problems to solve were randomly chosen among the ones presented in the different editions of the ACM-ICPC Word Finals, an annual multi-tiered competitive programming competition among the universities of the world [7].

These problems were browsed using the UVa Online Judge (UOJ) web, an online tool from Universidad de Valladolid which gathers these and others problems, as well as allowing the user to submit their solutions to them [8].

UOJ is helpful for us since it automatically validates the solution (it compiles and runs tests on the programs).

UOJ also measures programs execution time; however, we rather measured running time internally using *gettimeofday()* function, which returns monotonic time with µs granularity.

B. *Standard-compliant code performance*

After validating the code, we make it standard compliant using PRQA, a static analysis tool widely accepted in industrial code development. The tool is recognized as world leader in defect prevention, promoting safe coding practices and proactively ensuring the highest quality code for safety-critical and mission-critical systems [9].

PRQA comes with a variety of plugins for static analysis, including one specific for each of the two standards we are evaluating. The tool allows for code analysis, highlighting the source lines of code which have to be modified in order to be compliant with a given Secure Coding Standard.

Once the code is ready, it is validated and its performance is measured using UVa Online Judge once again.

C. *Statistical study*

We have performed a non-parametric test (one-tailed Mann-Whitney U test, significance level 0.10) to obtain the initial results.

We have first compared original code with SC-C code, and original code with M-C code. Then, we have compared M-C code with SC-C code.

Our hypothesis were: H0: *The difference of location between the samples is equal to 0*; H1: *The difference of location between the samples is lower than 0* (the former code running time is lower, that is, the program is faster).

IV. Experimental Testbed and Results

The following six ACM-ICPC problems were chosen and resolved: 136, 200, 494, 1056, 10035, and 10082 [8].

Code was written in ANSI C 5.3.0 and each test was repeated (each program was run and its running time was measured) a total of ten times for each version (regular/original, SC-C, and M-C). Later, Mann-Whitney U tests were performed (Table Tab. I summaries the results).

Table I
COMPARISON OF TWO SAMPLES (MANN-WHITNEY U TEST, ALPHA 0.10); RISK % TO REJECT THE NULL HYPOTHESIS H0 WHILE IT IS TRUE IS GIVEN BETWEEN BRACKETS

Program ref code in UOJ	Original faster than SC-C	Original faster than M-C	M-C faster than SC-C	SC-C faster than M-C
136	No (54.52)	No (82.77)	No (11.32)	No (90.07)
494	No (51.51)	No (23.63)	No (57.50)	No (45.48)
10035	Yes (3.78)	No (31.13)	No (71.49)	No (96.69)
1056	Yes (0.01)	No (13.65)	Yes (0.01)	No (100.00)
10082	Yes (4.45)	Yes (0.23)	No (50.00)	No (53.26)
200	No (54.51)	No (54.51)	No (66.12)	No (36.69)

Our prospective study suggests that both standards sometimes affect program performance and, when they do, they do it in a negative way (they make it slower). This happens more frequently with SC-C (3 out of 6 times) than with M-C (1 out of 6).

When comparing SC-C and M-S, the latter is found to be more optimal than the former 1 (almost 2, see program #136) out of 6 times, with no difference for the other cases.

V. Conclusions and Future Work

This research is a work in progress.

We have obtained some initial results showing that secure coding standards sometimes make the secured code slower (which is expected since they usually increase the number of source lines of code), and some results hinting that M-C may allow for faster code than SC-C in some cases.

However, choosing programs not complex enough (with too little lines of code to which only a reduced set of rules apply) may be somehow affecting these results.

In the near future, we will carefully select programs which source code is complex enough to be affected by as many standards rules as possible.

Also, smaller but topic-specific code fragments will be used, that is, code fragments which are functional and cover a given chapter or category of the standards (array usage, integer operations, string manipulation, and so on).

Acknowledgement

This research was partially supported by INCIBE (The Spanish National Cybersecurity Institute).

References

[1] "The 2016 Top Programming Languages - IEEE Spectrum", http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages [accessed: 2017/03/10]
[2] Graff, Mark, and Kenneth R. Van Wyk. "Secure coding: principles and practices". *O'Reilly Media, Inc.,* 2003.
[3] Seacord, Robert C. "Secure Coding in C and C++". *Pearson Education*, 2005.
[4] CMU SEI Cert Secure Coding Standard, https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard [accessed: 2017/03/10]
[5] CMU SEI Cert, http://www.cert.org/ [accessed: 2017/03/10]
[6] MISRA C, https://www.misra-c.com/Activities/MISRAC/tabid/160/Default.aspx[accessed:2017/03/10]
[7] The ACM-ICPC International Collegiate Programming Contest https://icpc.baylor.edu [accessed: 2017/03/10]
[8] UVa Online Judge https://uva.onlinejudge.org [accessed: 2017/03/10]
[9] PRQA, http://www.programmingresearch.com [accessed: 2017/03/10]

View publication stats