MISRA C, for Security's Sake![†]

Roberto Bagnara, *BUGSENG[*] and University of Parma[§] member[‡] of MISRA C Working Group, member[‡] of ISO/IEC JTC1/SC22/WG14 - C Standardization Working Group*

A third of United States new cellular subscriptions in Q1 2016 were for cars. There are now more than 112 million vehicles connected around the world. The percentage of new cars shipped with Internet connectivity is expected to rise from 13% in 2015 to 75% in 2020, and 98% of all vehicles are likely to be connected by 2025. Moreover, the news is often reporting about "white hat" hackers intruding on car software. For these reasons, security concerns in automotive and other industries have skyrocketed. We briefly illustrate the relationship between the ISO/IEC C language standards, the CERT C coding standards, ISO/IEC TS 17961 ("C Secure Coding Rules"), and MISRA C, with a focus on the objective of preventing security vulnerabilities (and of course safety hazards), as opposed to trying to eradicate them once they have been inserted in the code. We then introduce two new MISRA documents, MISRA C:2012 Addendum 2 and MISRA C:2012 Amendment 1, which clarify and complete the coverage offered by MISRA C:2012 against TS 17961. These new developments ensure that MISRA C, which is widely respected as a safety-related coding standard, is equally applicable as a security-related coding standard.

Joy and Pain of Connected Cars

Once upon a time, *embedded* systems were *isolated* systems. This is no longer the case, at least in the automotive industry. In the first quarter of 2016, connected cars accounted for around a third of all new cellular subscriptions, more than phones, more than tablets.[1] The percentage of new cars shipped with Internet connectivity is expected to increase rapidly, from 13% in 2015 to 75% in 2020 to 100% in 2025, when 98% of all vehicles will likely be connected. What happens to connected computers is well known and, according to the *Common Vulnerabilities and Exposures* (CVE) database, the situation is not improving despite the much increased awareness of security issues. For instance, *denial of service* vulnerabilities listed in the CVE have been increasing from 1999 up to and including 2016. The fear that car owners worldwide might become the target of various kinds of criminals is concrete. With the number of victims of ransomware running to hundreds of thousands and the consequent global losses likely to reach hundreds of millions of euros, it is conceivable that locking up cars until the owners pay might be seen as a profitable thing to do.

> "For well-organized attackers, this may end up being a numbers game, which may be similar to credit-card theft and sale."
>
> (Tony Lee, FireEye)

[†] Presented at the 14^{th} *Workshop on Automotive Software & Systems*, Milan, November 10, 2016.

[*] roberto.bagnara@bugseng.com

[§] bagnara@cs.unipr.it

[‡] Writing and speaking in a personal capacity.

[1] http://chetansharma.com/usmarketupdateq12016.htm

Safety and Security

The fact that the English words *safety* and *security* correspond to the same word in many languages is not helping to clarify the distinction between the two concepts. Commonly used taxonomies define *Safety* = *Integrity*+*Absence of catastrophic consequences* and *Security* = *Confidentiality* + *Integrity* + *Availability*, where *Integrity* can be described as the absence of improper (i.e., out-of-spec) system alterations under normal and exceptional conditions [1]. The only thing that distinguishes the role of integrity in safety and security is the notion of *exceptional condition*. This reflects the fact that exceptional conditions are perceived as accidental (safety hazards) or intentional (security threats) [3].

While safety and security are distinct concepts, when it comes to connected software not having one implies not having the other. To start with, both safety and security issues are due to software defects: some are introduced in requirements, some in design, some in coding. It is well known that many software defects that have an impact on safety, e.g., buffer overflows, can be exploited to attack a system if exposed to the outside world: we can summarize that by *Unsafe* + *Connected* $\implies$ *Insecure*. On the other hand, the recent successful attempts of Charlie Miller and Chris Valasek at impacting safety functions of the Jeep Cherokee show that *Insecure* $\implies$ *Unsafe*.

> "They want to drive trucks into civilians, and it's not too much to think they can hack a car and do the same thing."
>
> (John Carlin, U.S. Dept. of Justice)

C, CERT C, TS 17961, and MISRA C

ISO/IEC JTC1/SC22/WG14, a.k.a. the *C Standardization Working Group*, has always been faithful to the original spirit of the language [10]. With some little humor, this can be captured in the form of "commandments" like the following:

I Trust the programmer

II Let the programmer do anything

III Keep it fast, even if not portable

IV Keep it small and simple

Of course, some of these conflict with both safety and security requirements. All that is well known, as is well known that, for safety-related applications, language subsetting is crucial. The most authoritative language subset for the C programming language is MISRA C, now at its third edition [5], MISRA C:2012 or MC3 for short.

One of the unfounded myths about MISRA C is that it is only about safety. In reality, in its opening paragraph MISRA C presents itself as "a subset of the C language [that] can also be used to develop any application with high integrity or high reliability requirements." Because of this misunderstanding, when awareness of security threats increased, people started looking elsewhere for a security-related C coding standard, and they found *The CERT C (Secure) Coding Standard*, whose first edition was published in 2008 [8] and second edition in 2014 [9]. Both editions were authored by Robert C. Seacord when he was employed by the CERT Division of the Software Engineering Institute at CMU; Addison-Wesley owns the copyright [10].

Despite its popularity, CERT C has several shortcomings from an industrial point of view: it is the product of essentially one person (although a very expert and talented one, who, however, has now left the project); its development continues on a shared wiki that changes overnight;[2] it is not based on the idea of language subsetting, hence, differently from MISRA C, it leans on the *cure* side rather than the *prevention side*; many rules are formulated in a way that is not directly amenable to automatic, static analysis.

While working at what has become the C11 edition of the ISO/IEC C language standard, WG14 —which, starting from 2006, encouraged the development of CERT C— established, in 2009, a study group whose objective was to produce statically analyzable secure coding guidelines for the C language. The result of this effort has been the publication of the ISO/IEC TS 17961:2013 technical specification [4], TS 17961 in the sequel.

> "An essential element of secure coding in the C programming language is a set of well-documented and enforceable coding rules. The rules specified in this Technical Specification apply to analyzers, including static analysis tools and C language compiler vendors that wish to diagnose insecure code beyond the requirements of the language standard. All rules are meant to be enforceable by static analysis."
>
> (TS 17961, Introduction)

[2] A new "snapshot version" has been published in PDF form on June 30, 2016 [2].

MC3 for Safety *and* Security

As we saw, until and including the first quarter of 2016, the situation was the following: on the one hand, we had MISRA C:2012, which was and is widely respected as a safety-related coding standard (even though its prescriptions go beyond safety and are targeted at all high integrity and high reliability systems); on the other hand, we had TS 17961, a security-related coding standard backed by ISO. Even though, as mentioned, the software defects that can give rise to safety hazards and security threats have a significant intersection, MISRA C:2012 and TS 17961, as they were published in 2013, are not a substitute for one another.

Now, since the first quarter of 2016, things have changed. The MISRA C Working group has published two documents: the first is *MISRA C:2012 Addendum 2* [6], which contains a coverage matrix of MISRA C:2012 against TS 17961. The second document, *MISRA C:2012 Amendment 1*, contains 14 additional guidelines, 1 directive and 13 rules, targeted at the prevention of security issues [7].[3] These concern, with 'D' standing for *directive* and 'M'/'R' standing for *mandatory/required rule*, respectively:

1 D validation of external data;

1 M use of `sizeof()` on a function parameter of array type;

1 M `<ctype.h>` functions;

3 R `<stdlib.h>` memory comparison functions;

2 M `<stdlib.h>` environment functions;

2 M `<string.h>` string-handling functions;

1 R `<stdio.h>` I/O functions and handling of `EOF`;

3 R `<error.h>` handling of `errno`.

The coverage of MISRA C:2012, without (MC3) and with Amendment 1 (MC3 + MC3A1) is shown in Table 1. The coverage kind column has to be interpreted as follows [6]:

Explicit The behaviour addressed by the TS 17961 rule is *explicitly* covered by one or more MISRA C:2012 guidelines, which directly addresses the undesired behaviour.

Implicit The behaviour addressed by the TS 17961 rule is *implicitly* covered by one or more MISRA C:2012 guidelines, although the behaviour is not explicitly referenced.

[3] Both documents are available at `http://misra.org.uk/` .

Table 1: *MISRA C:2012 coverage of TS 17961*

Coverage kind	MC3	MC3 + MC3A1
Full, explicit	22	35
Full, implicit	7	3
Full, restrictive	11	8
Partial, broad	2	0
None	4	0
Total	46	46

Table 2: *MC3+MC3A1 coverage of CERT C*

Coverage kind	CERT C:2014	CERT C:2016
C11 specific	13	14
Full, explicit	41	42
Full, implicit	17	17
Full, restrictive	22	21
None	5	5
Total	98	99

Restrictive The behaviour addressed by the TS 17961 rule is covered by one or more MISRA C:2012 guidelines that prohibit a language feature in a *restrictive* manner. For example:

Rule 21.3 `stdlib.h`: memory alloc./dealloc.;

Rule 21.5 `signal.h`: all;

Rule 21.6 `stdio.h`: input/output functions;

Rule 21.8 `stdlib.h`: `getenv()`.

Broad Some aspects of the behaviour addressed by the TS 17961 rule are covered in a *restrictive* manner; some other aspects of the behaviour are not covered by any MISRA C:2012 guidelines.

None The behaviour addressed by the TS 17961 rule is not covered by any MISRA C guidelines.

Table 1 shows that, while coverage for freestanding applications was already very good before the issue of MISRA C:2012 Amendment 1, now coverage can be considered complete.

It is also interesting to see how MISRA C:2012 integrated with Amendment 1 covers CERT C. The MISRA C working group is currently working on the production of a document containing a proper compliance matrix. Preliminary, unofficial data can be seen in Table 2, both for the latest book edition, CERT C:2014 [9], and the PDF snapshot released on June 30, 2016 [2] (in which two rules have been added and one has been deleted).

There are other reasons why MISRA C:2012 with Amendment 1 is the best available coding standard for the development of critical embedded systems. One of them is the emphasis its guidelines put on readability: it is well known that code review combined with static analysis and the automatic enforcement of sound coding guidelines by means of high-quality tools is, by far, the most effective defect removal strategy. This is even more so when security concerns regarding, e.g., confidentiality or privilege escalation: while progress is being made on formal methods that are able to address them, careful code review is crucial both for today and for the foreseeable future.

> "In practice, then, security-critical and safety-critical code have the same requirements." (TS 17961, Introduction)

Conclusion

Connected cars are with us and in less than a decade they will be everywhere. Given the amount of software that equips them and the criticality of the functions controlled by it, connectivity opens the door to malicious activities of all kinds. This comes at a time when a redefinition of the concept of liability for producers of embedded software is taking place (cf. the Toyota unintended acceleration case). When it comes to security, additional issues arise. For instance, in order to demand a ransom, it is not even necessary to lock up or compromise safety of the vehicle: it is enough to, e.g., display on the dashboard something that makes the owner suspect safety might have been impacted by hackers. As another example, the security risk posed by disgruntled/unfaithful developers will likely affect the way software is produced and verified. The automotive industry, which already was the most critical sector for software safety, is also becoming to play the same role in regard to software security.

In this short essay, we recalled what makes safety and security different and what is common to them. We then reviewed the genesis of the CERT C coding standards and ISO/IEC TS 17961 "C Secure Coding Rules", and their relationship with the ISO/IEC C language standard. We discussed recent developments of the MISRA C:2012 coding guidelines, which complete the coverage of TS 17961. We argue that, with this integration, MISRA C:2012 is the C coding standard of choice for the automotive industry and for all industries developing embedded systems that are safety-critical and/or security-critical.

> "Car companies are finally realising that what they sell is just a big computer you sit in." (Kevin Tighe, Bugcrowd)

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[2] CERT. *SEI CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems*. Software Engineering Institute, Carnegie Mellon University, 2016 edition, 2016.

[3] K. M. Goertzel and L. Feldman. Software survivability: Where safety and security converge. In *AIAA Infotech@Aerospace Conference 2009*, Seattle, Washington, 2009. American Institute of Aeronautics and Astronautics.

[4] ISO/IEC. *ISO/IEC TS 17961:2013, Information technology — Programming languages, their environments & system software interfaces — C Secure Coding Rules*. ISO/IEC, Geneva, Switzerland, November 2013.

[5] MISRA. *MISRA-C:2012 — Guidelines for the use of the C language in critical systems*. MIRA Limited, Nuneaton, Warwickshire, UK, March 2013.

[6] MISRA. *MISRA C:2012 Addendum 2 — Coverage of MISRA C:2012 against ISO/IEC TS 17961:2013 "C Secure"*. HORIBA MIRA Limited, Nuneaton, Warwickshire, UK, April 2016.

[7] MISRA. *MISRA C:2012 Amendment 1 — Additional security guidelines for MISRA C:2012*. HORIBA MIRA Limited, Nuneaton, Warwickshire, UK, April 2016.

[8] R. C. Seacord. *The CERT C Secure Coding Standard*. Addison-Wesley, 2008.

[9] R. C. Seacord. *The CERT C Coding Standard: 98 Rules for Developing Safe, Reliable, and Secure Systems*. Addison-Wesley, second edition, 2014.

[10] R. C. Seacord. Safety and security coding standards for C. *Engineering & Technology Reference*, 2016. DOI: 10.1049/etr.2016.0024.