Secure embedded software – choosing the right coding standard

By Richard Bellairs, PRQA

This article compares state-of-the-art coding standards and explains how adherence to them can help developers deliver more secure C and C++ code.

An increasing number of products that touch our daily lives are connected to the internet. This brings many benefits, but also introduces potential security vulnerabilities. It is imperative that the software powering these products is developed with security in mind. Adoption of a strong coding standard that addresses known security issues has been shown to deliver more secure products. Enforcing this standard with automated code analysis tools will help to ensure products are delivered on time and in budget.

The pace of change in consumer products is amazing and the rate of innovation continues to accelerate. A new generation of connected devices and socially oriented services continues to impact our lives in profound ways, from the use of voice-activated speakers to control our smart homes to the hundreds of sensors that are used to control traffic in our cities more effectively. The widespread adoption of connected devices raises concerns over their security and our privacy.

Security of software is a hot topic, and one that every organization must address effectively. C remains the dominant language for embedded software development in consumer products, with C++ growing in popularity. There is no shortcut to achieving software security; reducing the risk it poses requires a concerted effort, and an appreciation of best practice industry guidelines. Applying coding standards to the development of safety-critical software is a widely adopted practice, but coding standards that target security issues are still relatively new. Demand for software security standards has increased as a result of the Internet of Things (IoT); the security of data and the connections between devices have been shown to have serious security flaws. Some examples of high-profile failures include the hacking of TrendNet nanny cams and the failure of Nest thermostats due to a flawed software update. Security and privacy breaches not only put users at risk, but also have the potential to cause significant damage to company reputation. As such, security is a commercial imperative.

Recognition of the importance of security has increased over recent years. New security-focused coding standards have emerged alongside the more mature safety-critical standards. Although the underlying goals are different, their recommendations frequently overlap. Most of the coding standards considered in this article use rules to prohibit aspects of a language that are considered inappropriate by the issuing standards body. In addition, they prescribe ways to enrich the development process and the language effectiveness. In some respects, they define a new language, with specific emphasis on delivering greater security, improved predictability, increased robustness and better maintainability. Today most popular coding standards for security are the CERT C Secure Coding Standard; MISRA C:2012; and the C Secure Coding Rules (ISO/IEC TS 17961:2013).

For the purposes of comparison, the coding standards covered in this article have been assessed using nine categories, some of which include a qualitative indication of how the coding standards perform. The performance indicator (1 to 3 stars) is derived from considerations and impressions PRQA has collected from its wide customer base which, by any measure, can be considered an official endorsement of any standard. We categorized the standards as follows:

Industry: the original industry sector targeted by the coding standard.

Reference language version: the version of the C Standard that is currently used as a reference for the coding guidelines. This is important as it can influence the choice of coding standard for a project; for example, if C11 is to be used (for instance, because some of its features make it the most applicable for a given application) MISRA C:2012 is not a good candidate unless specific compliancy requirements make it necessary.

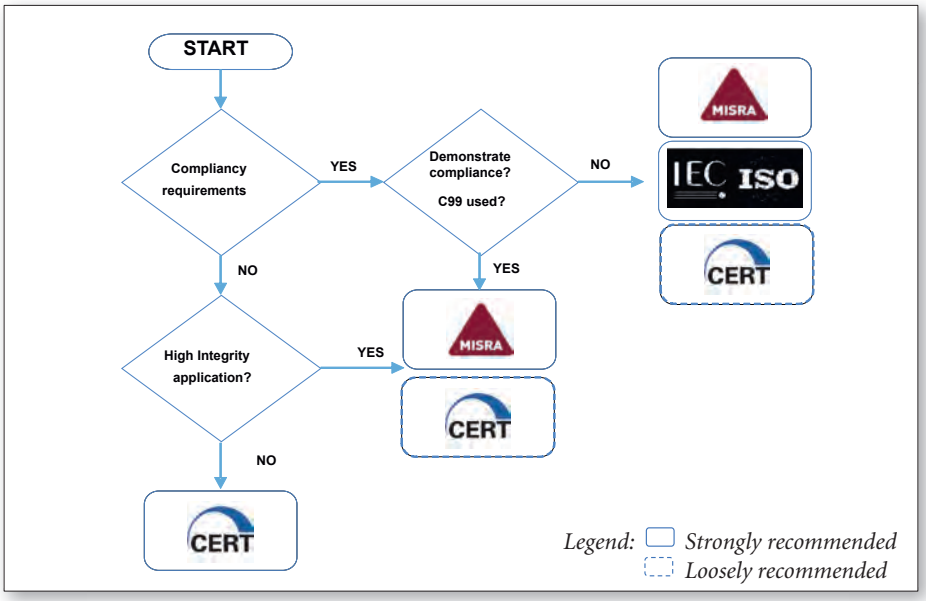

Figure 1. This simple flow chart helps to make an informed choice.

	CERT C	MISRA C:2012	ISO/IEC 17961:2013(E)
1 – Industry	Generic-agnostic	Embedded safety-critical applications	Generic-agnostic
2 - Reference Language Version	C11	C90, C99	C11
3 - Automatic Enforceability	★★	★★	★★★
4 – Coverage	★★★	★★	★
5 - Market Adoption	★★	★★★	★★
6 - Tool Availability	★★	★★★	★★
7 – Evolution	Wiki / Book		
8 – Resources	★★★	★★	★
9 – Examples	★★★	★★★	★★

Figure 3. Comparison table of CERT C, MISRA C: 2012, and IEC/ISO 17961:2013

Automatic enforceability: the ease of creation of automatic checks for the guidelines that don't result in false positives. This is usually related to how strictly or loosely specific guidelines are defined.

Coverage: a qualitative indication of the breadth of the coding standard scope and the number of guidelines defined. The broader the scope, the higher the educational value of the coding standard, but a broad scope can also bring complexity in terms of guideline maintenance and tool coverage.

Market adoption: the level of usage of the coding standard for real-world projects, in terms of formal compliance requirements (for example in functional safety applications) and voluntary usage to improve overall software quality.

Tool availability: the market availability of an automated code analysis tool to enforce the coding standard. This tends to be linked with the standards level of market adoption.

Evolution: a quickly evolving standard adapts better to feedback from the users and provides faster introduction of new features. This can be considered good for consumer products but bad for other sectors. CERT C uses two methods for publishing its guidelines; a web-based wiki, and a PDF document freely available. The wiki evolves faster than the latter.

Resources: this can include references to the C language standard, to other standards, papers, articles or other common knowledge bases that may be helpful.

Examples: descriptions to illustrate the issues related to the violation of a specific guideline and compliant solutions. The following page shows a comparison table covering the three featured coding standards and how they perform in the above categories.

There is no single best standard for secure coding. Selection must consider many different aspects, such as the duration of the project (where stability of the reference is more important), the version of the language being used and the existence of legacy code. The simple flow chart in figure 1 can help to make an informed choice.

Scenario 1. If the requirements dictate compliance with a recognized coding standard (a typical scenario would be a safety-critical application) then the choice should be MISRA C. The latest version of this standard is MISRA C:2012 Amendment 1. If a previous version of MISRA is mandated (for example MISRA C:2004) then the project will benefit from the addition of the security rules provided by ISO/IEC 17961:2013 (some work will be required to match the C version and remove any overlap – a good footprint would be the Addendum 2 of MISRA C:2012 "Coverage of MISRA C:2012 against ISO/IEC 17961:2012 C Secure").

Scenario 2. If the application has no compliancy requirements, and if there are no high-performance needs that would sacrifice code portability it is still recommended to adopt a high integrity perspective. In this case the recommendation would also be to use MISRA C:2012.

Scenario 3. In both previous scenarios CERT C could offer valuable support from a security perspective, and the recommendation would be to adopt CERT C in parallel with the suggested standards (in the flow chart this is indicated by a dashed line). However, if a high-integrity approach is not taken the MISRA C:2012 standard could be seen as too restrictive for the specific application, in this case the recommendation would be to only apply CERT C in order to achieve a good level of code security.

Choosing the right coding standard to adopt when developing secure code will depend on many factors, such as an understanding of the features and benefits of each standard and how they could meet the requirements of the current development project. The process shown in this article focuses on the ability to perform automated testing with tools such as PRQA static analyzers QA·C and QA·C++. Such tools perform deep analysis of software code to prevent, detect and eliminate defects and automatically enforce coding rules to ensure standards compliance. ■