Coding safe and secure applications

By Richard Bellairs, PRQA

Safety and security are different but there are some common ways to achieve them in high integrity applications, as this article explains.

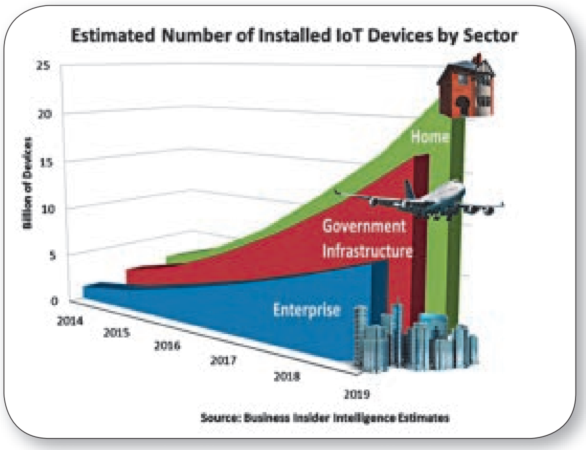

Figure 1. Estimated number of installed IoT devices by sector

■ The world is becoming far more connected, and systems are vulnerable to malicious attacks via these connections. There have already been some high-profile examples that are shaking the industry out of any complacency that may have existed. Safety in high integrity systems has long been a priority while security has not received the same focus, even though safety and security do have to meet different sets of rules and protocols. However, though they are intrinsically different, they do share some common themes and, because of this, when considering the coding aspects, it is possible to adopt a holistic approach. The need to address these concerns is present in every application, especially in safety critical systems; that said, it can be hard to provide a formal definition of what is secure and what is safe when talking about software development.

There are functional safety standards such as IEC61508 or ISO26262, but comparing the requirements in industry-recognised coding standards for high integrity systems with those in coding standards for security critical software, the common ground tends to expand. The discussion over safe and secure features of a language such as C or C++ is limited by the nature of the language itself; hence, what tends to emerge are styles and methodologies aimed at preserving safety and security in the application of one of more coding standards. The growth of interconnected devices able to provide advanced services, generally called the Internet of Things (IoT), is expected to expand exponentially in the next few years. While the promises of efficiency and cost reduction brought by this evolution are indeed attractive, with them come heavy concerns about security. The recent hack that allowed two researchers to take remote control of a modern SUV reverberated worldwide as a wake-up call for manufacturers and customers: if a technologically advanced system such as a modern high-end car can be subject to this kind of attack, what will happen to the most common, low-budget interconnected equipment that will represent the bulk of the many billion systems that will make up the IoT? Although the threat is well understood, the integration of security as a native element to drive development and business processes in a similar way to functional safety is still to come. This is far from reassuring given the number and level of risks presented by security vulnerabilities.

Ingraining a culture where processes that preserve security and safety coexist efficiently takes time and effort. The approach to be taken is by nature holistic and cannot be limited to single compartments or development stages. For example, the SUV hack exploited weaknesses and vulnerabilities at many different levels – architecture, service permission, password generation algorithms and so on. As a consequence, a sound product development process should integrate security hardening actions at all levels and allow them to coexist efficiently with the already demanding functional safety requirements. But what happens if the focus is only on software development? More specifically, what are the choices available to a developer with the task to programme a safety critical application and the need to be sure the application is also secure? Assuming sound measures have been applied at the requirements and design stages, it is time to choose how to translate them in efficient, secure, high integrity software.

Functional safety has two main families of reference standards dealing with software lifecycle organisation: IEC61508 and derived standards; DO178B/C and companion documents such as DO330. IEC61508 concerns functional safety of electrical, electronic and programmable electronic (EEPE) safety-related systems. It covers hazards caused by failure of the safety functions. Since it may be applied to any safety-related system that contains an EEPE device, its scope is pretty broad. Almost all major safety-related industry standards not connected with avionics are derived from IEC61508.

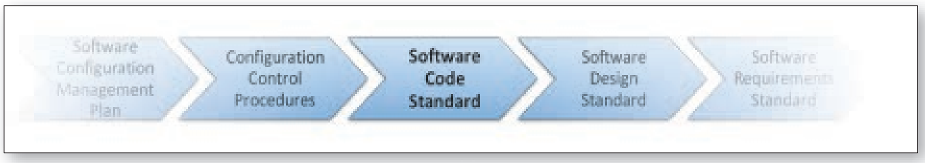
Software Configuration Management Plan
Configuration Control Procedures
Software Code Standard
Software Design Standard
Software Requirements Standard

Figure 2. Definition and documentation of the software development process

DO178C plus its companion documents DO330, DO331, DO332 and DO333 form the standards for airborne applications. DO178C is mandatory for any commercial avionics project looking to achieve FAA Certification. DO178C is more software-focused than IEC61508; the software safety level (or IDAL – item development assurance level) is determined from safety assessment and risk analysis and mapped on five levels, from A (catastrophic) to E (no effect).

For safety-critical applications, the definition of criticality of code has been widely analysed and there are standardised methods to qualify it and define proper ways to handle the development process. Safety integrity levels (SIL) in IEC61508, automotive SIL (ASIL) in ISO26262, software SIL (SSIL) in EN50128 or IDAL in DO178C are all examples of the same concept – to quantify the risk reduction required for a function according to the risk analysis and qualify the actions to be undertaken to assure this level is reached. Almost all industry-recognised functional safety standards prescribe the adoption of design and coding standards depending on the SIL targeted. Even if there is no authoritative indication of which coding standard is suitable for functional safety, one of the main references in this is MISRA C. ISO26262-6 acknowledges for the C language that MISRA C covers many methods required for software unit design and implementation, and its diffusion reaches all the major safety-critical applications, such as machinery, medical, nuclear power and railways. With DO178B/C, the situation is not so different. These standards require a thorough definition and documentation of the software development process. The base set of required documentation and lifecycle artefacts includes rich and detailed planning, and coding standard application is part of this list.

Coding standards such as MISRA define a subset of the target language. This avoids or limits usage of features and constructs that might lead to undefined or unspecified behaviour. Practices such as tolerating dead or unreachable code, which can cause issues when considering traceability and verification, are generally discouraged. Coding standards for high integrity applications tend to enforce features that produce a predictable behaviour. MISRA C: 2012, as an example, discourages the use of dynamic memory on the grounds that misuse of standard library facilities to manage dynamically allocated memory can lead to undefined behaviour. When the choice is made to use it, particular attention should be given to avoid unpredictable outcomes.

ISO/IEC27001: 2003 specifies the requirements for establishing, implementing, maintaining and continually improving an information security management system. It is based on the PDCA (plan, do, check, act) model, shared with all main management standards. Risk assessment and business impact analysis are used to identify and manage possible risks to the confidentiality, integrity and availability of information. A more detailed view on application security is taken by ISO/IEC27034: 2011, which provides guidance in defining and implementing information security controls by processes integrated in the system development lifecycle. As such, it's not a software application development standard, but does rely on existing standards. Moving towards security-oriented coding standards, the scenario is varied; it is possible to find secure coding standards for C and C++, as well as Java, Perl, PL/SQL and others.

There is a rich variety of techniques available to assess code security. Different issues can be tracked down using static analysis, dynamic and runtime evaluation, data flow and control flow tracking, taint analysis, executable analysis and heuristic analysis. These techniques can be effective and easy to implement depending on existing support for the selected language, built-in facilities, libraries and so on. The main reference point for security coding standards is CERT, which has for many years been publishing coding standards aimed at security enforcement. CERT coding standards are directly linked to real-world vulnerabilities found in the field by the common weaknesses enumeration (CWE). The CWE is a community-developed dictionary of software weakness types. The downloadable list of weaknesses can be navigated according to specific relationship contexts.

The CWE is related to a broader collection of publicly known information security vulnerabilities, known as CVE (common vulnerabilities and exposures), which is now the standard for common identification of vulnerabilities. CVE identifiers, also called CVE-IDs, provide reference points for data exchange for information security products and services. They are useful for analysing coverage and effectiveness of tools and services in regards to specific classes of vulnerabilities. NIST's national vulnerability database, the US government repository of standards-based vulnerability management data, contains more than 73,000 CVEs. MISRA C:2012 and CERT C can be considered champions of safety and security for the C language. Table 1 contains a quick comparison. There are noticeable differences between CERT and MISRA coding standards, but it's possible to define a strategy that results in the effective application of both on the same codebase.

Tools such as those available from PRQA are the most effective way to implement such a strategy. Such tools perform deep analysis of software code to prevent, detect and eliminate defects and automatically enforce coding rules to ensure standards compliance. They bring the added benefit of improved software maintainability and thus reduce overall development costs. Designing a safety-critical application while also optimising its security can be demanding. Safety and security require a set of strategies, processes, tools and skills that may not overlap, or may even conflict. Automated code analysis tools are an effective way to avoid coding defects that can lead to safety issues and security vulnerabilities as part of a holistic approach. ■