LDRA
Software Technology

Technical White Paper

Secure Code Properly

www.ldra.com

Secure connected embedded systems by shifting left

Connectivity has brought seismic change to embedded applications, with new opportunities for easier monitoring, upgrading, and enhancement. On the downside, connected systems present many more attack surfaces that are vulnerable to bad actors. Defending against such attacks is a daunting task, even for organizations well versed in functional safety. For those less familiar with the term, functional safety involves defending the world from a potentially flawed system, while cybersecurity defends a system from an imperfect world. Each presents its own unique challenges.

No single defence of a connected system can guarantee impenetrability; however, applying multiple levels of security ensures that if one level fails, others stand guard. Examples of such defence-in-depth approaches can include a mix of:

- Secure boot, which ensures the correct image loads
- Domain separation
- Multiple independent levels of security (MILS) design principles, such as least privilege
- Attack surface reduction
- Techniques for secure coding
- Security-focused testing such as static and dynamic analysis

Secure application code does little to defend a connected embedded system if the underlying architecture is unsecure. However, secure code has a key part to play in a system designed with security in mind. This remains true regardless of the preferred development life cycle design. Embedded development teams increasingly embrace DevOps principles, while others prefer the V model traditionally associated with functional safety standards such as DO-178C for aerospace, ISO 26262 for automotive, and IEC 62304 for medical devices.

From DevOps to DevSecOps for defence in depth

System connectivity has a big impact across all embedded software development, whether safety is critical or not. The days when a system could be developed, installed, and forgotten are long gone, and the need to respond to changing circumstances is paramount for many.

The DevOps approach owes much to lean manufacturing, which was designed specifically to be responsive to changing circumstances. Initially popular in enterprise system development, this approach brings clear benefits to many embedded applications. New market demands can be met faster through more integrated product development. Perhaps most important is the speed with which application patches and updates can be applied.

The need for updates is commonplace across industry sectors. Examples include Over-The-Air (OTA) security updates to automotive software and revisions to production parameters and processes in predictive maintenance in manufacturing plants.

DevSecOps expands on DevOps principles with a “shift left” principle—designing and testing for security early and continuously in each software iteration.

What is DevOps?

DevOps environments integrate development and operations teams to reduce silos. DevOps teams commonly use lean and agile principles to encourage collaboration and deliver software in a continuous integration / continuous delivery (CI/CD) practice that enables responsiveness to market opportunities and customer feedback.

Core principles in DevOps environments are to:

- Develop and test against production-like systems
- Deploy with repeatable, reliable processes
- Monitor and validate operational quality
- Amplify feedback loops
- Use automation to support collaboration, testing, and documentation

Defence-in-depth and the V-model

Traditionally, the practice for secure embedded code verification has been largely reactive. Code is developed in accordance with relatively loose guidelines and then subjected to performance, penetration, load, and functional testing to identify vulnerabilities that will be addressed later.

A more proactive approach ensures code is secure by design. That implies a systematic development process, where the code is written in accordance with secure coding standards, is traceable to security requirements, and is tested to demonstrate compliance with those requirements as development progresses.

This proactive approach integrates security-related best practices into the V-model software development life cycle that is familiar to developers in the functional safety domain. The resulting Secure Software Development Life Cycle (SSDLC) represents a shift left for security-focused application developers and provides a practical approach to ensuring that vulnerabilities are designed out of the system or addressed in a timely and thorough manner.

Although the context differs between DevSecOps and the SSDLC, shift left therefore implies the same thing for both—that is, an early and ongoing consideration of security.

Shift left in practice

The concepts embraced by the shift-left principle are familiar to individuals and teams developing safety-critical applications. For many years, functional safety standards demanded a similar approach. Consequently, many best practices proven in the functional safety domain apply to security-critical applications:

- Establish functional and security requirements at the outset or before each iteration
- Bidirectionally trace requirements to all stages of development
- Use a secure language subset
- Adhere to a security standard
- Automate processes for SAST (static) and DAST (dynamic) security testing
- Test early and often

Establish requirements at the outset

Undocumented requirements lead to miscommunication on all sides and create rework, changes, and bug fixes—as well as security vulnerabilities. To ensure smooth project development, all parts of the product and the process of its development must be understood by every team member in the same way. Clearly defined functional and security requirements help ensure that this is the case.

Such requirements are likely to define a complete system for V-model developers, and merely an iteration for those applying DevSecOps—but the principle remains the same. This is not to say that software can never be used as an "intellectual modeling clay" to create a proof of concept, but the ultimate result of such experimentation should be clearly defined requirements—and production code appropriately developed to fulfill them.

Provide bidirectional traceability

The IEEE Standard Glossary of Software Engineering Terminology defines traceability as "the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another." Bidirectional traceability means that traceability paths are maintained both forward and backward (Figure 11).

Automation makes it much easier to maintain traceability in a changing project environment.

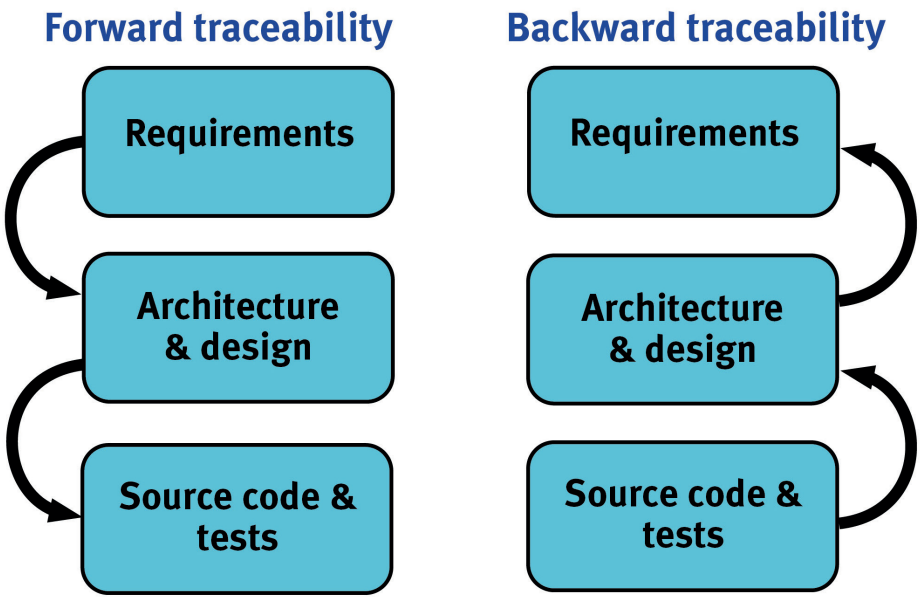

Figure 1: Bidirectional traceability.

Forward traceability demonstrates that all requirements are reflected at each stage of the development process, including implementation and test. The impact of any changes to requirements or of failed test cases can be assessed by applying impact analysis, which can then be addressed. The resulting implementation can then be retested to present evidence of continued adherence to the principles of bidirectional traceability.

Equally important is backward traceability, which highlights code that fulfills none of the specified requirements. Oversight, faulty logic, feature creep, and the insertion of malicious backdoor methods can all introduce security vulnerabilities or errors.

It is essential to remember that the life cycle of a secure embedded artifact continues until the last example in the field is no longer in use. Any compromise of such an artifact demands a response, a changed or new requirement, and one to which an immediate response is needed—often to source code that development engineers have not touched for a long time. In such circumstances, automated traceability can isolate what is needed and enable automatic testing of only the affected functions.

Use a secure language subset

For development in C or C++, research has long showed that roughly 80 percent of software defects stem from the incorrect usage of about 20 percent of the language. To address this, developers can use language subsets that improve both safety and security by disallowing problematic constructs.

Two common subsets are MISRA C and Carnegie Mellon Software Engineering Institute (SEI) CERT C, both of which help developers produce secure code. The two standards have similar goals but implement them differently. The biggest difference is the degree to which they demand adherence.

In general, development of new code with MISRA C results in fewer coding errors because it has stricter, more decidable rules that are defined in the basis of first principles. The ability to quickly and easily analyze software with reference to MISRA C coding standards can improve code quality and consistency and reduce time to deployment. By contrast, when developers need to apply rules to code retrospectively, CERT C may be a pragmatic choice. Analyzing code against CERT C identifies common programming errors behind most software security attacks.

Applying either MISRA C or CERT C results in more secure code. The manual enforcement of such standards on a code base of any significant size is not practical, so a static analysis tool is required to automate the job.

An overview of the differences between MISRA C and CERT C is shown in Table 1.

	MISRA C	CERT C
Uses	Designed for the development of embedded systems with high-integrity or high-reliability requirements. Used widely across critical sectors, including automotive, medical, avionics, railway, nuclear, telecommunications, IoT, and IIoT.	Often used for developing security-critical enterprise software with strict requirements, including web development.
Retrospective use	Should be adopted from the outset of a project. Retrospective adoption in an established, proven code base is likely to introduce problems.	Intended primarily to support new code development. A close second priority is remediation of old code.
Decidability and automation	Can be checked automatically with a static analysis tool. Results in portable code.	A tool cannot check the developer's understanding and implementation of many aspects of CERT C.

Table 1: Contrasting MISRA C and CERT C

Adhere to a security-focused process standard

Security-focused standards provide another piece of the secure development solution. In safety-critical sectors, such standards frequently complement those focused on functional safety. For example, J3061 "Cybersecurity Guidebook for Cyber-Physical Vehicle Systems" (soon to be superseded by ISO/SAE 21434 "Road vehicles – Cybersecurity engineering") complements the automotive ISO 26262 functional safety standard. Automated development tools can be integrated into developer workflows for security-critical systems and can accommodate functional safety demands concurrently, should the need arise.

Automate SAST (static) and DAST (dynamic) testing processes

Static analysis is a collective name for test regimes that involve the automated inspection of source code. By contrast, dynamic analysis involves the execution of some or all source code. The focus of such techniques on security issues results, respectively, in static analysis (or application) security testing (SAST) and dynamic analysis (or application) security testing.

There are wide variations within these groupings. For example, penetration, functional, and fuzz tests are all black-box DAST tests that do not require access to source code to fulfill their function. Black-box DAST tests complement white-box DAST tests, which include unit, integration, and system tests to reveal vulnerabilities in application source code through dynamic analysis.

Test early and often

All the security-related tools, tests, and techniques described here have a place in each life cycle model. In the V model, they are largely analogous and complementary to the processes usually associated with functional safety application development (Figure 2).

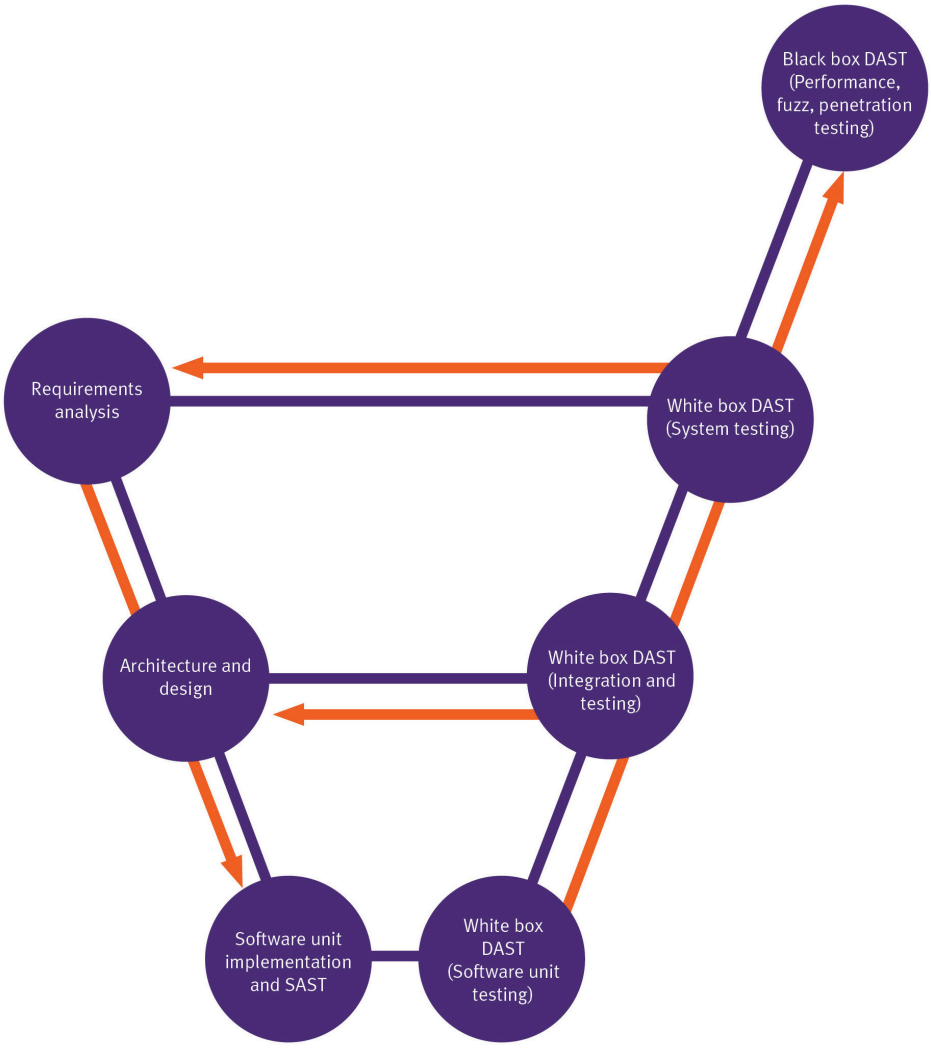

Figure 2: Use of security test tools and techniques in the V-model based secure software development life cycle (SSDLC)

In the DevSecOps model, the DevOps life cycle is superimposed with security-related activities throughout the continuous development process (Figure 3).

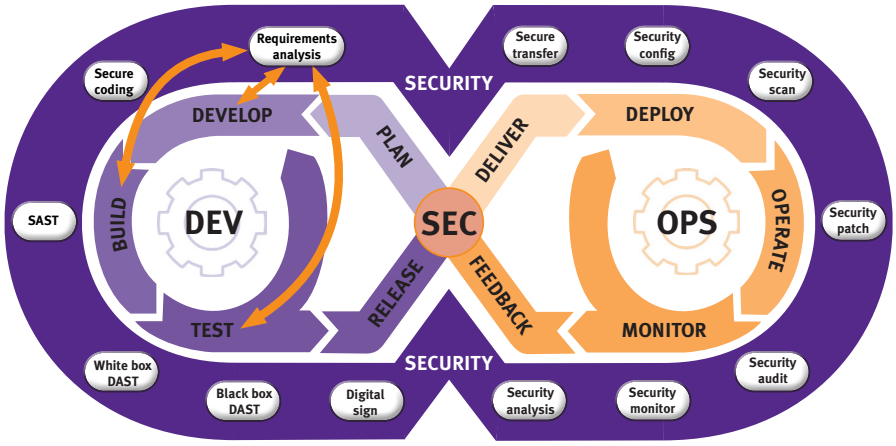

Figure 3: Use of security test tools and techniques in the DevSecOps process model[1]

[1]Based on extract from US DoD Enterprise DevSecOps Reference Design version 1.0 12 August 2019 https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf?ver=2019-09-26-115824-583

Requirements traceability is maintained throughout the development process in the case of the V-model, and for each development iteration in the case of the DevSecOps model (shown in orange in each figure).

Some SAST tools are used to confirm adherence to coding standards, ensure that complexity is kept to a minimum, and check that code is maintainable. Others are used to check for security vulnerabilities but only to the extent that such checks are possible on source code without the context of an execution environment.

White-box DAST enables compiled and executed code to be tested in the development environment or, better still, on the target hardware. Code coverage facilitates confirmation that all security and other requirements are fulfilled by the code, and that all code fulfils one or more requirements. These checks can even go to the level of object code if the criticality of the system requires it.

Robustness testing can be used within the unit test environment to help demonstrate that specific functions are resilient, whether in isolation on in the context of their call tree.

Fuzz and penetration black-box testing techniques traditionally associated with software security remain of considerable value, but in this context are used to confirm and demonstrate the robustness of a system designed and developed with a foundation of security.

Automated development tools pave the way to security

Automated development tools are a critical element in creating secure embedded software. Developers should have access to automated software tools to support their work through the entire software development life cycle from traceability and engineering to static and dynamic software analysis to unit and integration testing. The accuracy, determinism, and formal reporting capabilities of these tools meet the assurance requirements for the development of reliable, security-critical software.

Automated tools are critical elements of a secure embedded software development environment, whether as part of a V-model (SSDLC) or to support lean and agile processes and CI/CD practices (DevSecOps). An entire team, including project managers, systems engineers, developers, QA managers, and test and maintenance engineers can leverage these tools throughout the software development life cycle.

Develop secure application code with LDRA tools

LDRA offers automated software testing tools to expedite development, certification and approval processes to:

- Trace all requirements, design, and verification artifacts throughout the software development life cycle
- Demonstrate compliance with coding standards (SAST)
- Automate unit test and system-level test (white-box DAST)
- Perform and report coverage analysis on all code down to the target level
- Plan for and execute requirements-based testing
- Integrate with other software across the tool chain
- Produce software certification and approval evidence automatically

For more than 40 years, LDRA has developed and driven the market for software that automates code analysis and software testing for safety-, mission-, security-, and business-critical markets. Working with clients to achieve early error identification and elimination and full compliance with industry standards, LDRA traces requirements through static and dynamic analysis to unit testing and verification for a wide variety of hardware and software platforms. LDRA products are designed to be used on safety- and security-critical applications, so they require high-quality assurance. The company's Quality Management System has been ISO 9001 certified since 1993, and LDRA products were first TÜV approved in 2013.

LDRA is among the first companies to support updates and new international guidelines and standards that ensure the safety and security of software-based electronics systems. LDRA offers sound advice to the industry, including consultation on the development of security-critical application code. Over 40 years of experience informs the development of tools and the provision of consulting services at LDRA and the company's growing reputation as a training provider.

LDRA actively participates on standards groups and closely monitors security standards such as:

- **ISO/SAE 21434** – LDRA will be among the first to support the automotive cybersecurity standard ISO/SAE 21434 when its contents are confirmed.
- **CWE** – LDRA has achieved Common Weakness Enumeration (CWE) compatibility for the LDRA tool suite.
- **CERT** – The LDRA static analysis secure coding standards tools for CERT C, CERT C++, and CERT Java automate the security software development life cycle.
- **MISRA** – The LDRA tool suite automates source code checking for conformance to any version of the MISRA rules (MISRA C:2012, MISRA C++:2008, MISRA Compliance:2020).

A complete compliance matrix is available for every coding standard supported by LDRA, giving complete transparency of which rules are implemented within the tools. This matrix is designed to simplify comparison of tool compliance to multiple versions of each standard and to assess compliance with multiple standards.

Boasting a worldwide presence, LDRA has headquarters in the United Kingdom, United States, Germany, and India coupled with an extensive distributor network.

For more information on the LDRA tool suite, visit www.ldra.com.

www.ldra.com

LDRA UK & Worldwide
Portside, Monks Ferry,
Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

LDRA Technology Inc.
2540 King Arthur Blvd, Suite 228,
Lewisville, Texas 75056
United States
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd.
Unit No B-3, 3rd Floor Tower B,
Golden Enclave, HAL Airport Road,
Bengaluru
560017
India
Tel: +91 80 4080 8707
e-mail: india@ldra.com

Secure Code Properly White Paper v 2.0 08/24