Ultimate Guide to Embedded Systems Security

What is embedded systems security?

01/

Embedded systems security provides mechanisms to protect an embedded system from all types of malicious behavior. In this section, you'll learn about embedded systems security, related security terms, software and physical security, and four qualities of embedded systems that affect security.

Definition of embedded systems security

Embedded systems security is a cybersecurity field focused on preventing malicious access to and use of embedded systems. Embedded systems security provides mechanisms to protect a system from all types of malicious behavior. Cybersecurity specialists work with systems design teams to ensure the embedded system has the necessary security mechanisms in place to mitigate the damage from these attacks.

Embedded systems security focuses on preventing malicious access to and use of embedded systems.

Cybersecurity terms

Let's look at some general cybersecurity terms that are helpful to know as you learn about embedded system security:

- An **attack vector** is a path an attacker or malicious process could take to compromise a system. Flash drives, the Internet, network protocols and disks are embedded systems attack vectors.
- An **attack surface** is a target point of exposure, or the end goal of the attack vector. For example, a network driver, a user application and a file system are attack surfaces.
- A **threat actor** is a source (software or a person) with malicious intent.
- An **attacker** is an individual performing a malicious act in real time.
- A **vulnerability** is a weakness that can be exploited by a threat actor to perform unauthorized actions within an embedded system or computer.

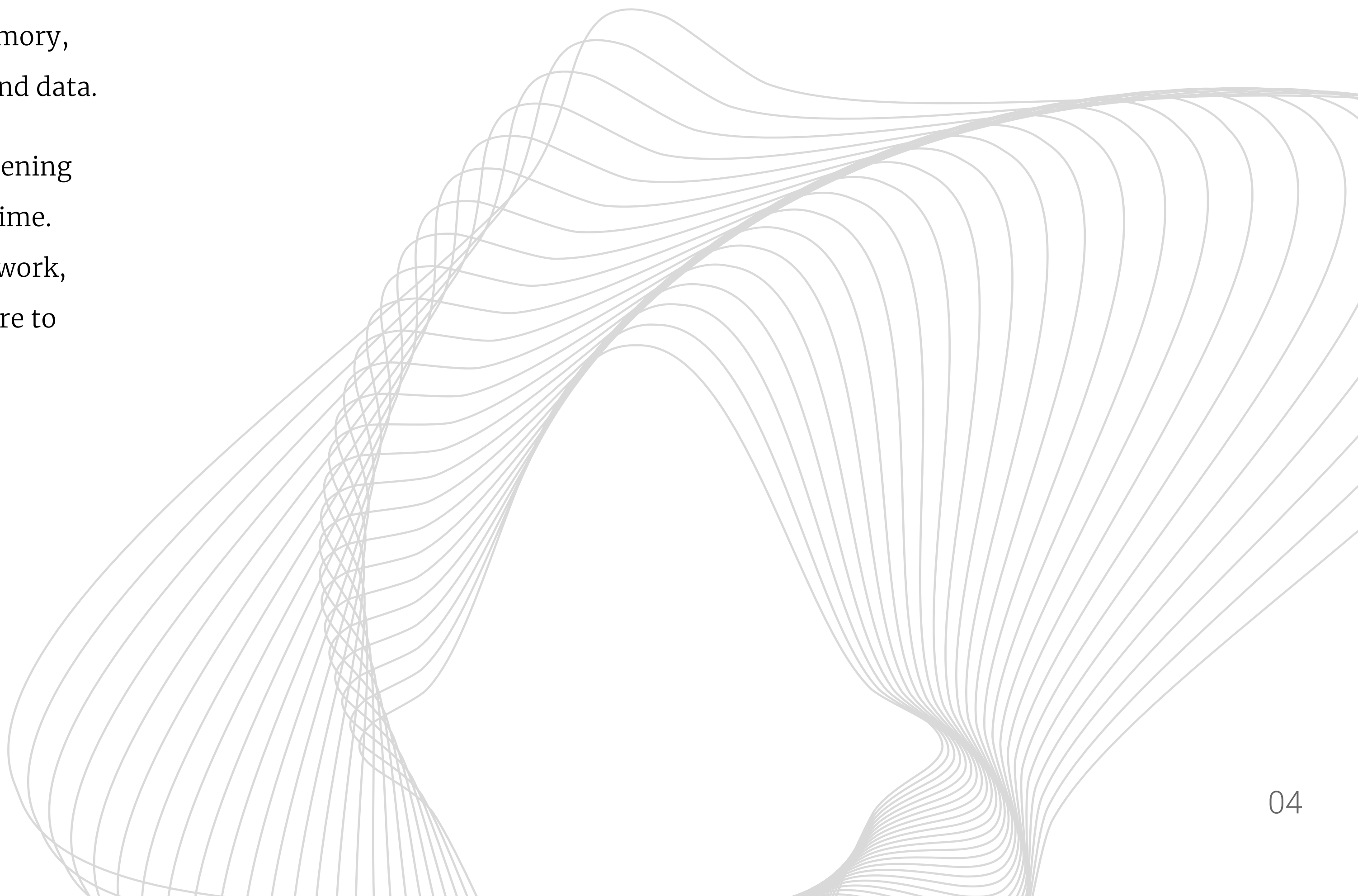
Software security vs. physical security for embedded systems

Two types of security apply to embedded systems: physical security and software security.

- **Physical security**, such as locked doors and surveillance cameras, keeps an unauthorized person present on location from accessing an embedded system, physically damaging it or stealing it. Physical security limits access to sensitive areas and equipment. Physical security may also include attributes of a device itself, including immutable memory technology, such as eFuses to store secure bootloader keys, tamper-resistant memory, protected key stores and security enclaves to protect sensitive code and data.
- **Software security** manages and responds to malicious behavior happening in the system, both during the initialization process and during run time. Software security features include authentication of a device to a network, firewalling network traffic and stringent hardening of system software to name a few.

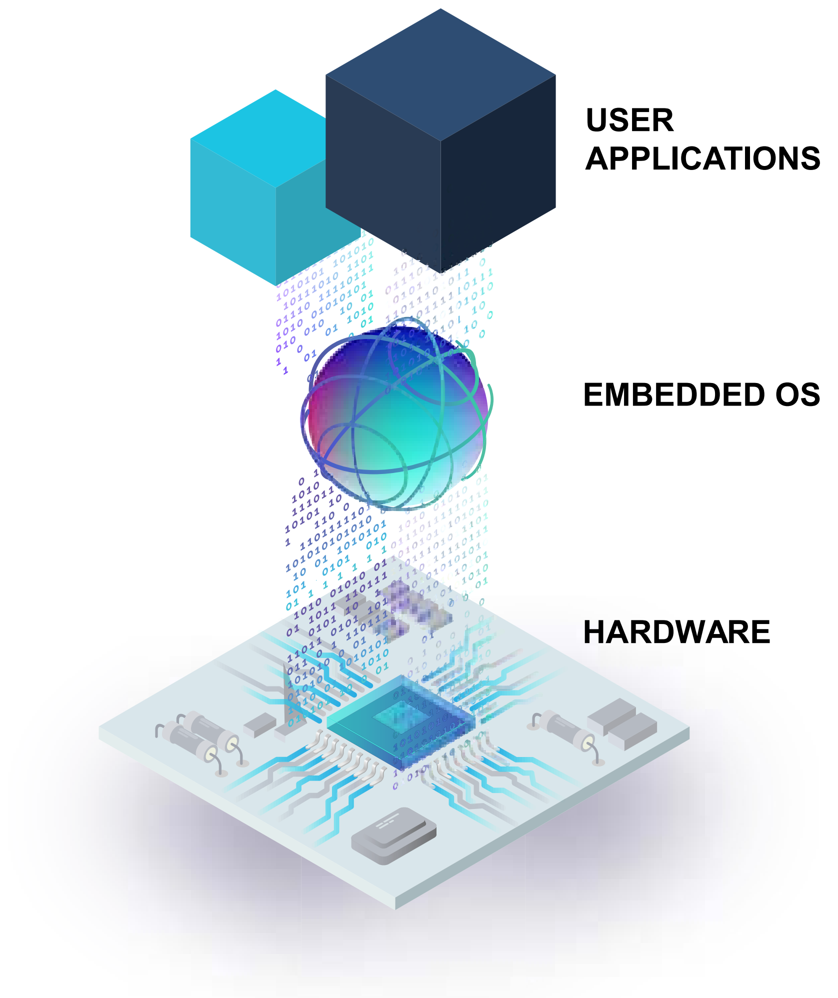
Qualities of embedded systems that affect security

Many embedded systems perform mission-critical or safety-critical functions vital to a system’s intended function and surrounding environment. Embedded systems security is relevant to all industries, from aerospace and defense to household appliances. Modern embedded systems are starting to become interconnected by the Internet of Things (IoT), which creates additional attack vectors.



Connected systems

The most secure embedded system is one that is turned off, and the next most secure system is completely isolated. When embedded systems were islands of technology that contained minimal information, embedded software security was less important. Embedded systems are now often connected to a communications network that exposes the system to more threat actors.



Cyberattack targets

The monetary value of data, the ability to cause serious harm, and the interoperability and connectivity of modern embedded systems, including mission-critical systems, make embedded systems popular targets. Cyberattacks on embedded systems range from disabling vehicle anti-theft devices and degrading the performance of control systems to directing printers to send copies of documents to the hacker and accessing a smartphone's data. Cyberattacks on embedded systems create an urgent need for everyone from developers to end users to help prevent, manage and patch vulnerabilities.

All elements of the hardware and software architecture need to be secure. Each of the components of embedded system architecture creates an attack surface, from the firmware and embedded operating system (OS) to middleware and user applications. The embedded OS, a foundational piece of embedded systems security, plays the leading role as the backbone of security for an embedded system.

Figure 1:

The operating system is the security foundation for embedded systems and can help prevent exploits in both the underlying hardware and applications/middleware.

Cyberattacks on embedded systems create an urgent need for everyone from developers to end users to help prevent, manage and patch vulnerabilities.

Product lifecycle

Some embedded systems are in the field for decades, others for just a few years. Many mission-critical systems, such as cars, defense systems and power plants, have a long service life — 20 years or more. Older embedded systems often don't get updated because the hardware is obsolete and doesn't support the new software. Designing a system to be secure can greatly increase the viability of keeping systems safely in service and at reduced risk of attack.

Developers need to consider hardware and software obsolescence when designing embedded systems to increase system longevity and security. Computing, networking, cyberattacks and embedded systems security will evolve over the lifespan of an embedded system in ways that cannot be foreseen by system developers. As vulnerabilities are identified, they will need to be mitigated with patches, which require software updates. Including security in the design phase helps ensure that an embedded system has a way to get updates and is capable of running new software.

Difficult to update

Some embedded systems are easier to update than others. A smart TV or smartphone can be updated regularly with minimal impact to the end user. In comparison, insecure software in a modern vehicle can put lives at risk, so software updates to vehicles are carefully orchestrated (and costly).

The type of embedded OS also affects the update process and frequency. Applying an update to an embedded system running a monolithic OS, such as Linux, is difficult. When the OS and all OS services run in kernel space, applying an OS service patch requires a full OS install, OS refresh, and a full system reboot— all of which increase the scope of testing and the time to deploy.

In comparison, the architecture of a microkernel OS, such as the QNX® Neutrino® real-time operating system (RTOS), makes embedded software updates much easier. OS services in a microkernel run outside of kernel space, which allows for the rebooting of a single service, without a kernel reboot, resulting in very minimal impact on kernel behavior. In addition, the footprint of a microkernel OS service update is generally small – it doesn't necessarily require the kernel to be updated at the same time – reducing the time and cost of testing a patch.

02/ Security vulnerabilities and exploits

It’s good to understand your hidden enemies and their tactics. Some embedded system attacks are active: they change the behavior of the system. Other attacks are passive: they read data and spy. In this section, you’ll learn about the anatomy of an embedded system exploit, four attack paths, the most common vulnerabilities and why some OS present a larger attack surface than others.

Anatomy of an embedded system exploit

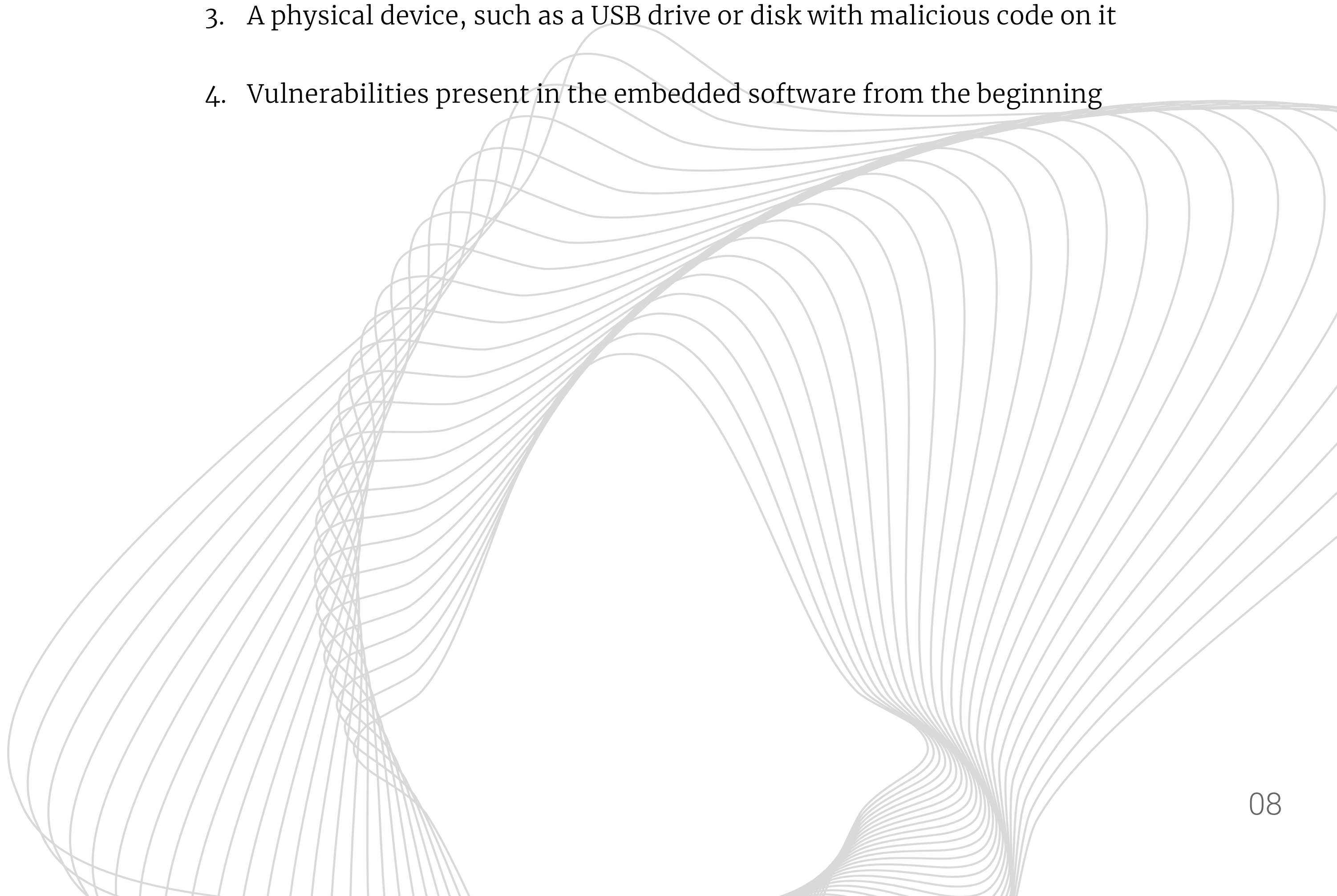
How does a threat actor exploit an embedded system? In general, most cyberattacks follow these five steps:

1. Get network (or physical) access
2. Understand the underlying processes, hardware and embedded operating system (reconnaissance)
3. Find a vulnerability in the host-based protection, such as within a programmable logic controller (PLC), embedded OS or middleware
4. Manipulate the controller
5. Exploit the vulnerability to attack the system or others

Embedded system attack vectors

A threat actor gains control of an embedded system through one of these four paths:

1. The larger system (the host) that includes the embedded system
2. The Internet or a communications network that connects the embedded system with other devices
3. A physical device, such as a USB drive or disk with malicious code on it
4. Vulnerabilities present in the embedded software from the beginning



Embedded software vulnerabilities

Like computers, many embedded systems have security vulnerabilities that can provide a way for a threat actor to gain access to the system. Typically, there is a time lag between the discovery of a specific vulnerability — such as a CVE, misconfiguration, or weak or missing encryption — and the availability and application of a patch or other remediation. Meanwhile, vulnerable systems are at risk. System hardening and the use of additional layers of security — such as a managed security service, firewall or intrusion detection and prevention system (IDPS) — reduce the risk that a threat actor will successfully exploit the vulnerability.

The five most common types of software vulnerabilities in embedded systems are as follows:

Buffer overflow

Buffer overflow attacks occur when a threat actor writes data or code to a memory buffer, overruns the buffer's limits and starts overwriting adjacent memory addresses. If the application uses the new data or new executable code, the threat actor may be able to take control of the system or cause it to crash.

Improper input validation

If an embedded system requires user input, a malicious user or process may provide unexpected input that causes an application to crash, consume too many resources, reveal confidential data or execute a malicious command. The unexpected input could be a negative value, no input at all, a path name outside of a restricted directory, or special characters that change the flow of the program.

Improper authentication

Authentication proves users and processes are who they say they are. Improper authentication may allow a threat actor to bypass authentication, repeatedly try to guess a password, use stolen credentials or change a password with a weak password-recovery mechanism.

Information exposure

Many types of sensitive information may be exposed to a threat actor directly or to another party. Information exposure could reveal information ranging from personal data to business secrets and from system logs to message headers. Data spoofing and device hijacking are two of the ways threat actors expose sensitive information.

Improper restriction of operations within the bounds of a memory buffer

If the programming language or the embedded OS do not restrict a program from directly accessing memory locations that are outside the intended boundary of the memory buffer, a threat actor may be able to take control of the system or cause it to crash, much like a buffer overflow attack.

As many as 20 percent of industrial control systems have critical security issues.

03/ Best practices for embedded security

Embedded systems security must be addressed in a holistic manner with best practices throughout the software development life cycle. In this section, you'll learn about the security advantages of a microkernel OS, the hardware-software partnership, essential defense mechanisms and mitigations of common embedded security challenges, cybersecurity standards for embedded systems developers and more ways to secure embedded systems throughout the software development lifecycle.

Security advantages of microkernel OS

A microkernel OS is structured with a tiny kernel space with services like file systems provided in user space, drivers or network stacks. Less code running in kernel space reduces the attack surface and increases security. The microkernel works with a team of optional cooperating processes that run outside kernel space (in the user space) and provide higher-level OS functionality. Only the core OS kernel is granted access to the entire system, and component isolation prevents an error in one component from affecting other parts of the system. A determined hacker, as long as they don't have root access, can only crash one component at a time when the system runs a microkernel OS or a secure embedded hypervisor.

The microkernel design and modular architecture of the QNX Neutrino Real-time Operating System enables BlackBerry® QNX® customers to create compelling, safe, and secure devices built on a highly reliable OS, a software foundation that helps guard against system malfunctions, malware and security breaches.

Hardware technologies provide a root of trust and encryption and decryption services.

Embedded security is a hardware-software partnership

No matter how advanced and security-aware, software alone cannot ensure embedded systems security. Hardware, software and cloud vendors must work together. For example, hardware technologies ensure device boot integrity, and on-chip security capabilities enable robust key management and encryption, which is too computation-intensive for embedded software alone. Hardware capabilities enable the OS to provided security functionality, such as access control policies, encrypted file systems, rootless execution, path space control and thread-level anomaly detection.

Root of trust

A hardware security module (HSM) or hardware root of trust manages keys, performs encryption and decryption functions, and embeds keys for OS and application use. Often these system-on-a-chip (SoC) components provide CPU offload for bulk encryption and decryption, and they may also be used to offload network cryptographic functions.

Hardware roots of trust are increasingly available as part of the SoC but can also be integrated using discreet electronics, such as an authentication IC or a TPM. During manufacturing, a private key can be generated on a chip or injected into each chip to serve as a root of trust. When the private key is certified by a public key infrastructure (PKI), the secure device identifier can become a foundational component of trusted device connectivity. For example, a secure device identifier can help a developer or designer establish trusted communications with peer devices and cloud-based services.

Secure boot

Secure boot leverages the signature provided by a device trust anchor, the public part of the root of PKI used to sign device code. When the embedded system boots, the boot image will be validated using this public key and the corresponding trust chain to ensure that boot-time software has not been tampered with. Establishing the provenance of the original software and of any software updates typically relies on digital signatures from a public key cryptosystem. But in some instances a hybrid model can be used. In a hybrid model, symmetric key cryptography is used to validate software integrity and speed the boot code verification process for time-critical startup requirements. Unlike code verified with a public key, the symmetric key must remain secret, known only to the device.

Trusted execution environment

A trusted execution environment (TEE) or hardware security zone provides hardware-enforced isolation in a secure area built into the main processor, which allows the software developer to establish a device root of trust. A TEE may run in a secure mode of the processor (e.g., ARM TrustZone) or on a separated, isolated CPU core that acts as a security co-processor to the SoC. TEEs typically allow trusted applications to perform security-critical processing on behalf of the embedded system.

Trusted platform module:

A trusted platform module (TPM) provides hardware-based security functions such as a cryptoprocessor to generate, store and use internal cryptographic keys; encryption of keys and other sensitive material stored in device memory; and measurement and attestation of the integrity of a system state during the boot process.

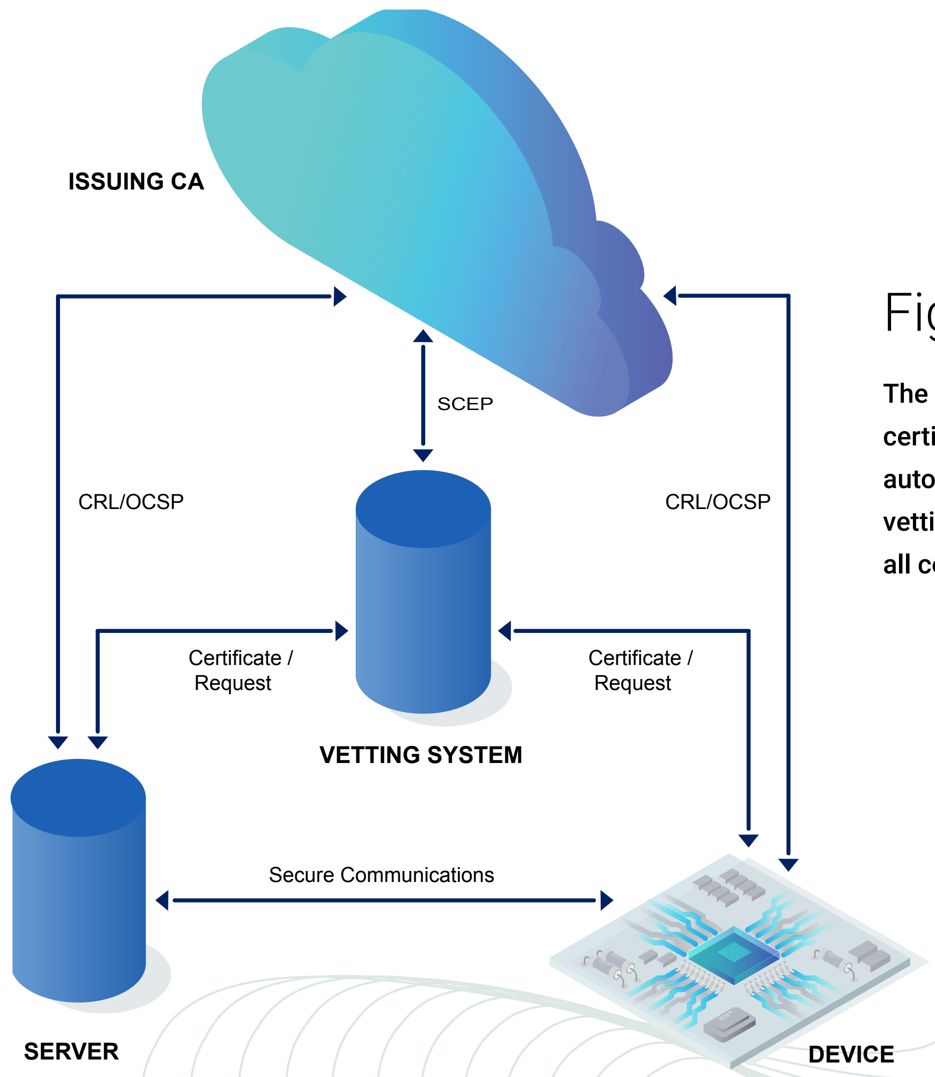


Figure 2:

The process of issuing certificates can be automated with a trusted vetting system verifying all certificate requests.

Essential defense mechanisms of a secure OS

Embedded OS can be made more secure with additional defense mechanisms. For example, memory corruption via buffer overflow is a common vulnerability in embedded software. Three examples of software security techniques—executable space protection (ESP), address space layout randomization (ASLR), and stack canaries—can help the OS defend against exploits. These three capabilities, which we at BlackBerry® QNX® call the Three Musketeers, are security basics that every OS should have. Let’s look at each one:

Executable space protection

Executable space protection (ESP) marks specific memory regions as non-executable, so that an attempt to execute machine code in those regions causes an exception.

Address space layout randomization

Address space layout randomization (ASLR) allocates the base address of the stack, heap and shared memory regions to new locations every time a new process is executed, making buffer overflow attacks difficult because a threat actor can’t predict where the information will be stored.

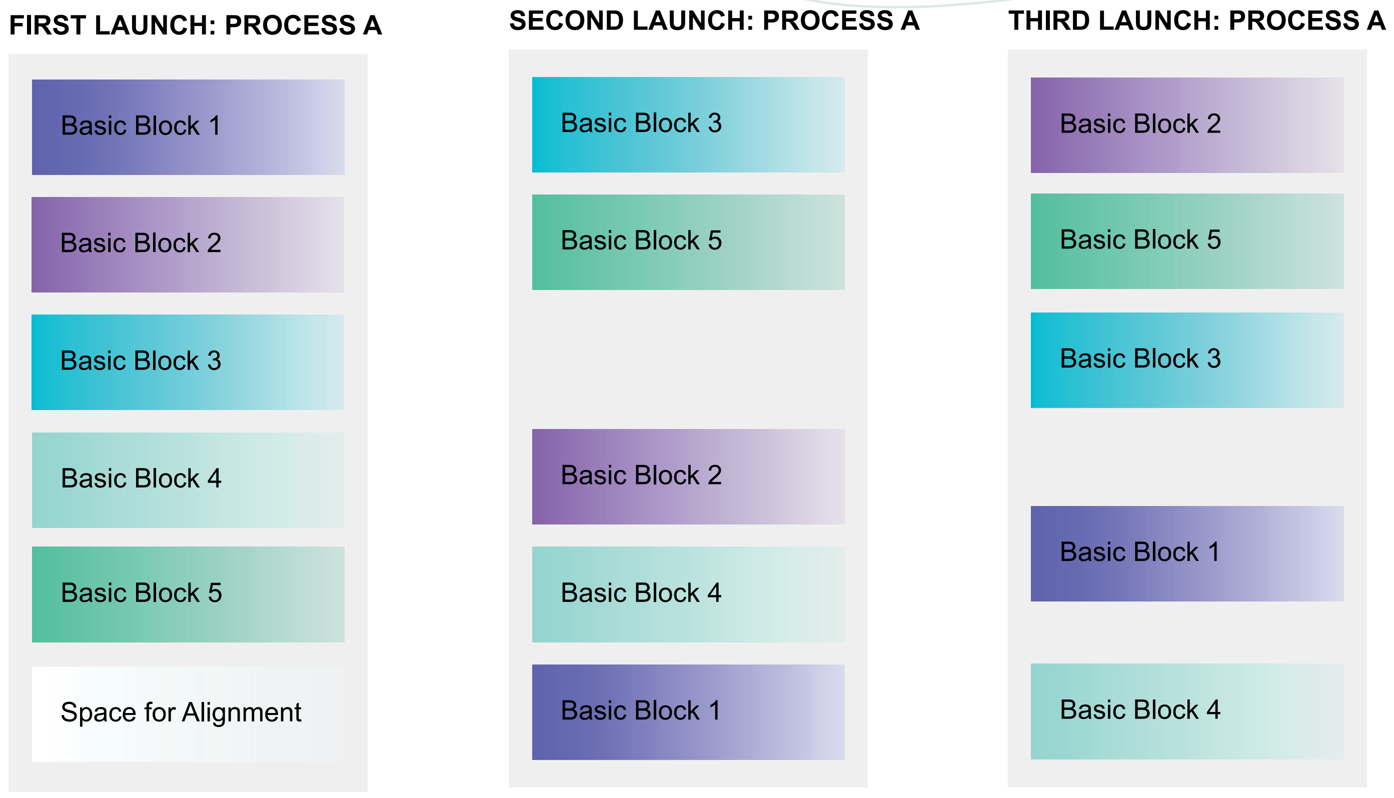


Figure 3:

Address space layout randomization (ASLR) randomly positions the binary code and data every time the system executes a new process, making it difficult for threat actor to guess where they are stored in memory.

Stack Canaries

Stack canaries allow the operating system to detect a stack buffer overflow before executing malicious code. The OS places a small random integer before the stack return pointer and checks for it before overwriting memory. If the stack value has changed, the OS will stop execution and cause an exception.

Although 70 percent of the most popular embedded OS lack at least one of these defense mechanisms, QNX RTOS provides all three.

Read more about the Three Musketeers and other risk mitigation techniques in our white paper, The Past, Present and Future of Cybersecurity for Embedded Systems.

A secure OS can help solve common embedded security challenges

Security plays a critical role in keeping embedded systems protected against threat actors. Security needs to be considered at all stages of embedded system design instead of being bolted on at later stages. The design and security features of an embedded OS built for embedded security can allow it to defend and protect itself against a wide variety of attacks and help you mitigate security challenges, such as the following:

Challenge	Goal	Mitigations
Data Confidentiality at Rest	Prevent a threat actor from seeing, modifying, or exfiltrating sensitive data on the system	File-based encryption
Data Integrity	Package system assets in an integrity-protected container that can be mounted at any time on the system for access.	Mark all executables as non-writable Use an integrity-protected disk solution Secure boot
Unrestricted Access to System Resource Managers	Prevent unauthorized system components from accessing system resource manager channels or restrict the operations they can request after they connect	Security policies POSIX permissions and access control lists (ACLs)

Challenge	Goal	Mitigations
Filesystem Object Access Control	Restrict access to filesystem objects by various processes	POSIX permissions and access control lists (ACLs)
Untrusted Code Execution	Prevent a threat actor from running or loading an untrusted binary from a filesystem	Use a utility to mark files and filesystems as trusted
Redirect Control Flow	Prevent a threat actor from modifying executable control flow	Use a compiler to mark the relocation sections of an executable as read-only
Repeatability of Attacks	Make it harder for an attacker to guess where code is loaded in memory	Address space layout randomization (ASLR)
Buffer Overflows	Instrument code to mitigate potential buffer overflow attacks	Use a compiler with fortified function support to detect out-of-bounds memory accesses
Stack Overflows	Instrument code to mitigate stack overflow attacks	Compile code with stack canaries
Revealing Sensitive System Information	Prevent a threat actor from being able to inspect the private information of other processes on the system	Secure the /proc filesystem Security policies
Process Execution with Least Privileges	Configure the system to restrict privileges on processes to make sure that they execute with the least privileges necessary for their tasks	Security policies Use a granular system of policies to govern which operations a particular process is permitted to do POSIX permissions and access control lists (ACLs)
Device Hardware Access to Kernel Memory	Prevent a threat actor from using a device (such as a GPU or network card) to directly access memory (DMA) addresses to which the device has not been explicitly granted access	Use a system memory management unit manager that uses DMA containment
Denial of Service	Prevent a threat actor from interfering with critical systems by exhausting system resources	Set resource limits, such as setrlimit() system calls in C

Cybersecurity standards for embedded systems

In addition to using a secure software foundation, security standards provide best practice processes to help developers build secure embedded systems and conform to cybersecurity regulations. While functional safety standards for embedded systems are mature, standards for embedded systems cybersecurity are not. Currently, the automotive industry is leading the way with two publications, SAE J3061 and ISO/SAE 21434, and the WP.29 regulation that goes into effect in January 2021. The following resources provide embedded developers with expert guidance:

- **UNECE WP.29 Regulation on Cyber Security and Software Update Processes:** This international regulation establishes performance and audit requirements for new passenger vehicles in many countries with a goal of paving the way for connected and autonomous vehicles.
- **SAE J3061, "Cybersecurity Guidebook for Cyber-Physical Vehicle Systems:"** The publication provides a cybersecurity process framework and guidance to help organizations identify and assess cybersecurity threats and design cybersecurity into cyber-physical vehicle systems throughout the entire development life cycle process.
- **ISO/SAE 21434, "Road Vehicles – Cybersecurity Engineering:"** A work-in-progress, ISO/SAE Draft International Standard (DIS) 21434 will likely be published in 2020 or 2021. The standard will be help manufacturers demonstrate compliance with WP.29 regulation and is likely to be adapted for use in other industries.

Embedded systems developers can also find security guidance in these two cybersecurity frameworks:

- **ISO/IEC 27001:2013, "Information technology — Security techniques — Information security management systems — Requirements"** can help any organization establish, implement, maintain and continually improve an information security management system. ISO 27001 also includes requirements for the assessment and treatment of information security risks.
- **U.S. National Institute for Standards and Technology (NIST) "Cybersecurity Framework:"** First published in 2014 as the "Framework for Improving Critical Infrastructure Cybersecurity", the CSF is widely considered the first line of cybersecurity defense across all industry sectors.

Risk assessments lead to security requirements

In addition to security techniques and security standards, the secure software development lifecycle (secure SDLC or SSDLC) can help every developer build more secure systems. Embedded system design should always begin with an analysis of the device and its intended and potential unintended usage, security risks (attack vectors) and attack surfaces. Security should also be considered at every stage in the SDLC process.

The first step in the SSDLC is a thorough risk assessment, which will inform the security requirements. A risk assessment identifies threats, the likelihood of those threats and the damage they can cause.

In addition to the risk assessment, threat models provide a structured approach to identifying and characterize threats to enable a more secure system design. Two popular threat models are STRIDE and DREAD.

The secure SDLC can help every developer build more secure embedded systems.

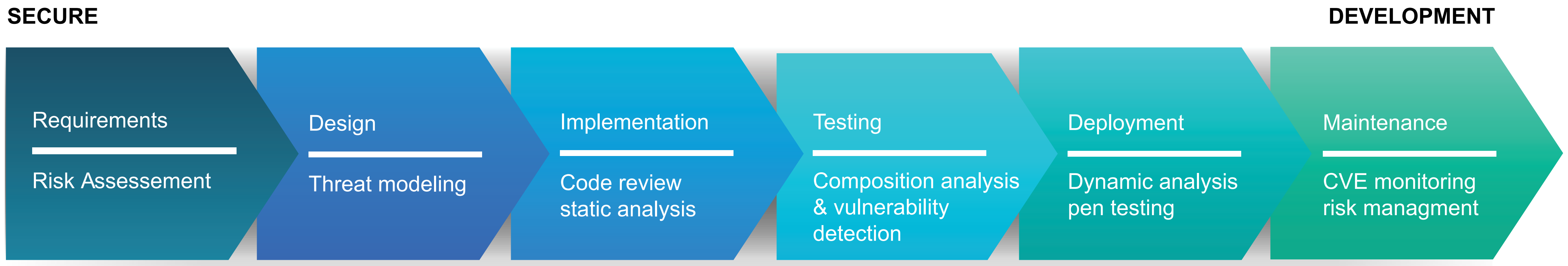

SECURE
DEVELOPMENT
Requirements
Risk Assessement
Design
Threat modeling
Implementation
Code review static analysis
Testing
Composition analysis & vulnerability detection
Deployment
Dynamic analysis pen testing
Maintenance
CVE monitoring risk managment

Figure 4:

Example of a secure software development lifecycle

Supply chain security

A common security requirement is the systematic verification of the security of software and hardware components in the internal and external supply chain. A trusted components program includes requirements such as:

- A trusted system components supplier program should identify suppliers with mature secure development life cycle processes, such as a security-aware manufacturing process to mitigate the risk of malware entering the supply chain.
- A system-on-a-chip (SoC) should support device provenance and integrity (e.g., secure boot and OS loading) through the use of immutable trust anchors, which authenticate access to internal debug interfaces to ensure on-chip security functionality cannot be bypassed. SoCs should support hardware root of trust capabilities, such as system integrity validation, secure key storage, and operations such as encryption, decryption and digital signatures. If a trusted execution environment (TEE) hardware root of trust is not available, the design should, at minimum, provide a device with a unique encryption key that cannot be easily extracted or revealed.
- Device software should be of known and trusted provenance, from integrity-protected and verifiable archives. Many off-the-shelf components are considered software of unknown provenance, or SOUP.
- Sensitive keys and device identifiers should not be exposed in any part of the manufacturing process, and there should be a clear audit trail of any provisioning process.
- Records of device manufacturing should include traceability to both hardware and software bill of materials, so that potential component defects can be identified when the system is deployed. Device identifiers should be cryptographically secured with a system that can detect counterfeit, grey-market and remanufactured components on the original production line and during system repair.
-

Embedded security encryption

Information exposure is a common vulnerability in connected systems, so embedded security often involves encryption. Transport layer security (TLS) can thwart information exposure attacks, including data spoofing and device hijacking.

How TLS works

1. After a connection is made between a client and server, the client requests a secure connection and tells the server what types of cryptographic security the client supports.
2. The server chooses the most secure option supported by both the server and client, and then sends a security certificate signed with the server's public key.
3. The client authenticates the server's certificate, generates a secret key encrypted with the server's public key and sends the encrypted key back to the server.
4. The client and server use the secret key to generate a pair of symmetric keys (or two pairs of public-private keys) and communication commences securely.

Trusted communication

All communications between modules and between the embedded system and the outside world should be authenticated, trusted and encrypted. Each connected device should have its own unique private key and certified device identifier. This device certificate allows each device to authenticate to a cloud directly or via a separate security gateway to enforce security policies.

Additionally, all network traffic should be authenticated and encrypted with rolling keys. Device certificates can be used to support client authentication. This is an increasingly common way to prevent the impersonation of IoT devices and support secure peer-to-peer connectivity.

Protecting encrypted data

Securing sensitive device data, such as user data or proprietary information, is also critical. Only a user or device with authorization should be able to decode the encrypted data. This means sensitive keying material needs to be protected, such as by personalizing embedded devices with their own unique hardware keys or using hardware key stores or integrity protection modules (IPM). It is also a best practice to allow only privileged/authorized processes in a trusted state to have access to OS-level or application key stores

Code review and testing

During the implementation and testing phase of the secure SDLC, code review and testing is a required step. However, it is important to note that code testing of source code or binary alone will not ensure the system is secure. It will identify security related software bugs, but it will not identify system wide or process related security issues.

SAST vs. DAST vs. penetration testing

Static application security testing (SAST), dynamic application security testing (DAST) and penetration testing are three types of software testing that identify vulnerabilities. These types of security testing can also find unnecessary services (FTP, SSH) and open ports that expose attack surfaces.

- **SAST** tools look inside the code to identify common security flaws, such as buffer overflows and cross-site scripting vulnerabilities, without running the code.
- **DAST** tools, such as vulnerability scanners, run on operating code to find vulnerabilities such as code injection and authentication errors.
- In **penetration testing,** an ethical hacker (also called a white hat) attempts to break in to ascertain if a determined attacker could gain access or disrupt the embedded system. Penetration testing is also called pen testing or pentesting.

Binary code analysis

Although most SAST is performed on source code, BlackBerry® Jarvis™ is an example of a binary code analysis tool. Jarvis scans binary files included in a build and provides metrics and cautions that tell an embedded software developer what to improve to reduce the security debt of the code, which is the accumulation of outstanding tasks that have relevance to security. (Security debt is a term used in agile development as part of secure agile software craftsmanship.)

Threat defense and in-field tests

Once the device is in the field, intrusion detection and intrusion protection systems (IDPS) intercept communications defensively to identify or block attacks and the exfiltration of data. Some embedded systems security services, such as BlackBerry Cylance, take a proactive approach through threat hunting and security monitoring of embedded systems and IoT devices.

In addition, self-tests assess the security posture of an embedded system in the field. Software for self-testing analytics and diagnostics monitors events, logs crashes and anomalies, and sends this information to the cloud. A cloud-based system can then analyze the information and act to mitigate safety and security risks.

Secure OTA updates over the lifetime of a system

Maintenance is the final phase in a secure SDLC. What do you do when you find a vulnerability in software after the product ships? Updating embedded systems once they are out in the world is much more difficult than updating software on personal devices, such as laptop computers. Just identifying the physical location of embedded systems and their status (e.g., software version, in service) can be difficult. So, updating software for safety- or mission-critical systems — where downtime or restarts can have a catastrophic impact — must be performed with the utmost care and only after extensive testing of the impact on the whole system.

Until recently most embedded software updates were performed in person, which is incredibly costly and resource intensive. In recent years, a number of over-the-air (OTA) update solutions have emerged for embedded systems. But because embedded system infrastructure components (e.g., authentication mechanisms, end-point management systems, cloud, software repository, communication protocols) don’t often interoperate, out-of-the-box solutions rarely work.

BlackBerry QNX has created a flexible OTA solution that can be customized to seamlessly combine existing technology and manage even the most complex update scenarios.

How BlackBerry QNX can help you

04/

BlackBerry QNX is trusted in critical systems globally to provide the software foundation for safe, secure and reliable systems. In this section, you'll learn more about our secure software solutions, professional security services and security industry leadership.

The QNX Neutrino RTOS and the QNX Hypervisor are the gold standard operating systems for secure embedded software.

Secure software foundation

BlackBerry's reputation as a security vendor is backed by 40 years of experience delivering secure and reliable software for embedded systems. BlackBerry offers the world's most trusted mobile security, and the QNX Neutrino RTOS and the QNX Hypervisor are the gold standard in embedded software operating systems and hypervisors. BlackBerry QNX has the expertise needed to help customers build more secure products.

BlackBerry QNX provides businesses in the most demanding industries with the building blocks for secure embedded systems; these include:

- A secure embedded OS with a microkernel architecture and adaptive partitioning
- A layered approach to security through mechanisms such as secure boot, integrity measurement, sandboxing, access controls and rootless execution
- A comprehensive security policy that enables system architects and integrators to control access to specific system resources and determine the type of access permitted (e.g., no root access)
- Secure over-the-air (OTA) updates for software maintenance over the lifetime of an embedded system

BlackBerry Security Services

Leveraging BlackBerry's industry-leading cybersecurity expertise, we can evaluate your software assets to identify vulnerabilities and recommend specific remediation actions. From penetration testing to a holistic appraisal of your company's security posture, our security and embedded system experts can assess and address security issues with your processes or products at every stage of your software development life cycle (SDLC). We can help you:

- Assess your current software security posture and goals
- Plan and implement a quantifiable security strategy
- Improve your level of software security assurance
- Choose the best security solutions and services vendors

Our security services include:

- Cybersecurity security and regulation readiness
- Open source software (OSS) assessment
- BlackBerry Jarvis software security services
- Software security audit (also called vulnerability assessment)
- Penetration testing
- Security training and workshops
- Custom engagements ranging from complex multi-system integration projects to ongoing staff augmentation support

Industry leadership

BlackBerry Advanced Technology Development Labs (BlackBerry Labs) works at the forefront of research and development in the cybersecurity space. With a strong focus on data science and machine learning, BlackBerry Labs' innovation funnel investigates, incubates and facilitates technologies specifically designed to further our commitment to safety, security and data privacy for BlackBerry customers.

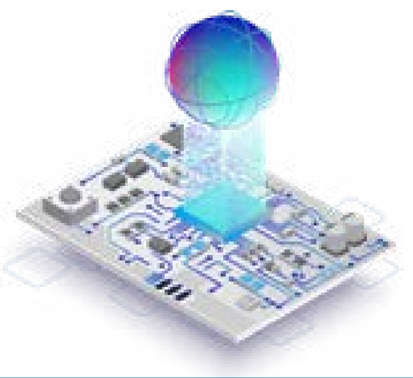
BlackBerry Certicom provides device security, anti-counterfeiting, and product authentication to deliver end-to-end security with managed public key infrastructure, code signing and other applied cryptography and key management solutions.

BlackBerry Cylance is an AI-based endpoint security solution that prevents breaches and provides added controls for safeguarding against sophisticated threats. Focusing on a stronger prevention-based approach versus signature-based prevention tools, BlackBerry Cylance has redefined what an endpoint protection solution can and should do.

BlackBerry QNX offers the most advanced and secure embedded operating system (OS) and embedded hypervisor for mission-critical and safety-critical embedded systems.

Check Out Our Other Ultimate Guides

Autonomous Systems



Real-Time Operating System

Functional Safety and Safety Certification

Contact Us

Our team of experts are here to answer your questions

About BlackBerry® QNX®

BlackBerry QNX is a leading supplier of safe, secure, and trusted operating systems, middleware, development tools, and engineering services for mission-critical embedded systems. BlackBerry QNX helps customers develop and deliver complex and connected next-generation systems on time. BlackBerry QNX technology is trusted in over 175 million vehicles and more than a hundred million embedded systems in medical, industrial automation, energy, and defense and aerospace markets. Founded in 1980, BlackBerry QNX is headquartered in Ottawa, Canada.