



## **Cinnabar Networks**

*A Division of Bell Security Solutions Inc.*

# **Security as a Core Competency of the QNX Neutrino Microkernel**

**Eugen Bacic**  
**Director, Emerging Technology**  
**Bell Security Solutions Inc.**

# Security as a Core Competency of the QNX Neutrino Microkernel

## TABLE OF CONTENTS

### Introduction 3

### The Foundations of Computer Security 3

The Principles of the Reference Monitor 3

Defining Security Within a System 4

Modern Computer Security Foundations 5

### Security: A Foundation of the QNX Neutrino Microkernel 5

Microkernel Architecture 5

Anderson's Three Principles 6

Tamper Resistant 7

Always Invoked 7

Small and Simple 7

Salter & Schroeder's Eight Principles 8

Economy of Mechanism 8

Fail-Safe Defaults 9

Complete Mediation 9

Open Design 10

Separation of Privilege 10

Least Privilege 11

Least Common mechanism (Resource Protection) 11

Psychological Acceptability 13

### Modern Enhancements to Security 13

Accountability 13

Priority of subjects and priority of operations 14

Self-tests 14

Fault Tolerance 15

### Conclusion 15

## 1 INTRODUCTION

Secure systems. This phrase invokes the thought of a system that is so locked down that it is difficult to use, or that users shy away from using. Security has, in fact, become synonymous with “user unfriendly.” However, this isn’t the case when security is properly understood and well implemented.

In this paper, we illustrate what is understood and expected for a secure system and the necessary practices required to implement secure systems. We will demonstrate how the QNX<sup>®</sup> Neutrino<sup>®</sup> realtime operating system (RTOS) addresses security requirements while meeting user expectations. We will approach the subject from the point of view of the security community, from an understanding of system security that has been developed through years of experience and through analysis of all manner of systems, large and small.

## 2 THE FOUNDATIONS OF COMPUTER SECURITY

The foundations of computer security can be traced back to *Project MAC* and *Multics*, the original security work performed under the guidance of Roger Schell of the National Computer Security Center (NCSC). The *Multics* work led directly to the development of the *Orange Book*<sup>1</sup> and many of the founding principles of computer security.

Chief among these founding principles were the notions laid out by James Anderson in 1972 and by Jerome Saltzer and Michael Schroeder in 1974. These principles still form the basis of good computer security and the very foundations of the various evaluation criteria, from the *Orange Book* through the ITSEC<sup>2</sup> and CTCPEC<sup>3</sup> to the *Common Criteria* today.

### 2.1 THE PRINCIPLES OF THE REFERENCE MONITOR

In 1972 Jim Anderson defined security in his seminal *Computer Security Technology Planning Study*<sup>4</sup> as consisting of three basic principles:

---

<sup>1</sup> The *Trusted Computer Security Evaluation Criteria*, published in 1983 and updated in 1985, was colloquially known as the *Orange Book* due to its bright orange cover. It was superseded by the short-lived *US Federal Criteria* and ultimately by the internationally recognized *Common Criteria*. For more information visit <http://en.wikipedia.org/wiki/TCSEC>.

<sup>2</sup> The *Information Technology Security Evaluation Criteria (ITSEC)* was created by France, Germany, the Netherlands, and the United Kingdom and based on their collective criteria work. It focused primarily on assurance.

<sup>3</sup> The *Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)* provided the first *modular* evaluation criteria allowing for the creation of *profiles* that define functionality against which an evaluation would occur. Its ideas were incorporated into the *Common Criteria*.

<sup>4</sup> Visit the NIST’s collection of early computer security papers at <http://csrc.nist.gov/publications/history/>.

- Tamper resistant
- Always invoked
- Small enough to be subject to analysis and tests to ensure that it is correct

Anderson refined these principles into a single, constituent mechanism that he termed the *Reference Monitor*. As can be seen in the study, these principles are also the fundamental components of good software engineering and, by extension, good kernel development. In fact, modern security experts equate “Reference Monitor” with the core security enforcement mechanisms within the kernel. Good development practices and security are more synonymous than most people believe.

The role of this fundamental security-enforcement mechanism is to validate all access attempts made to resources (data, peripherals, programs, etc.) by any given process. Hence, Anderson defined how security within a kernel ensures that a resource is accessed not only by the appropriate process but also by the right process operating against the correct data in the correct context.

As will be seen, the QNX Neutrino operating system’s kernel implementation of the POSIX standard and its strict use of the memory management unit (MMU) captures the functionality required to meet the fundamental definition of security as defined by Anderson’s Reference Monitor.

## 2.2 DEFINING SECURITY WITHIN A SYSTEM

In 1974, Jerome Saltzer and Michael Schroeder expanded upon Anderson’s work, defining what has come to be known as *The Eight Principles of a Security Design* in their paper, *The Protection of Information in Computer Systems*.<sup>5</sup>

|                             |  |
|-----------------------------|--|
| <b>Economy of mechanism</b> | Reduction of complexity to eliminate unexpected side effects or behavior   |
| <b>Fail-safe defaults</b>   | Access must be explicitly given and implicitly denied                      |
| <b>Complete mediation</b>   | An always-invoked, consistent mechanism that protects designated resources |
| <b>Open design</b>          | No reliance on security through obscurity                                  |

---

<sup>5</sup> To download the University of Virginia’s digital version of the paper, visit <http://www.cs.virginia.edu/~evans/cs551/saltzer/>.

|                                    |   |
|------------------------------------|---|
| <b>Separation of duties</b>        | Multi-factor verification for access                        |
| <b>Least privilege</b>             | Access rights provided to perform a given task, but no more |
| <b>Least common mechanism</b>      | Resources aren't shared implicitly                          |
| <b>Psychological acceptability</b> | The system must be understandable by users.                 |

Although Saltzer and Schroeder were working from the perspective of defining security within operating systems, they had in fact described what today is commonly referred to as “good programming practices” and “good software design practices.”

### 2.3 MODERN COMPUTER SECURITY FOUNDATIONS

By combining what Anderson, Saltzer, and Schroeder defined in the early 1970s into a single thesis of security, today's security community has defined evaluation criteria, risk, and vulnerability assessment methodologies, along with many other formal methods of determining the security posture of an application, system, or network.

This whitepaper provides a brief discussion of how QNX Software Systems, through its own strict methodologies and functional requirements for the QNX Neutrino RTOS, adheres to the principles defined by Anderson, Saltzer, and Schroeder that form the basis of modern computer security.

## 3 SECURITY: A FOUNDATION OF THE QNX NEUTRINO MICROKERNEL

### 3.1 MICROKERNEL ARCHITECTURE

In a microkernel OS, the kernel provides a minimal set of primitives, or system calls, to implement the basic necessities of an operating system; see Figure 1. These basic necessities typically include address-space management, thread management, and interprocess communication. All other services normally provided by traditional, or *monolithic*, kernels are implemented in *user space* as individual processes or programs.

By virtue of its small, comprehensible size, a microkernel is less prone to errors, provides a superior security platform, and allows for easier validation of implementation than conventional operating system kernels. Also, because system services (networking, device drivers, file systems) are implemented outside of the kernel as separate, user-space processes, a failure in any service won't impact the performance of the kernel or of other running processes. This modular approach also provides for both flexibility and ease of extensibility.

Microkernel architecture makes for a more secure and safe system since the actual *security-aware* component — the microkernel — is small and easily understood with a focus on what has been historically defined as *security relevant*. For example, because the microkernel demands that all access to system services go through its own well-defined interfaces, it can actually enforce security. And the fact that these services are independent, and can be shown as such, provides not only enhanced maintainability, but also enhanced security and reliability.

The QNX Neutrino RTOS is based on one such microkernel.

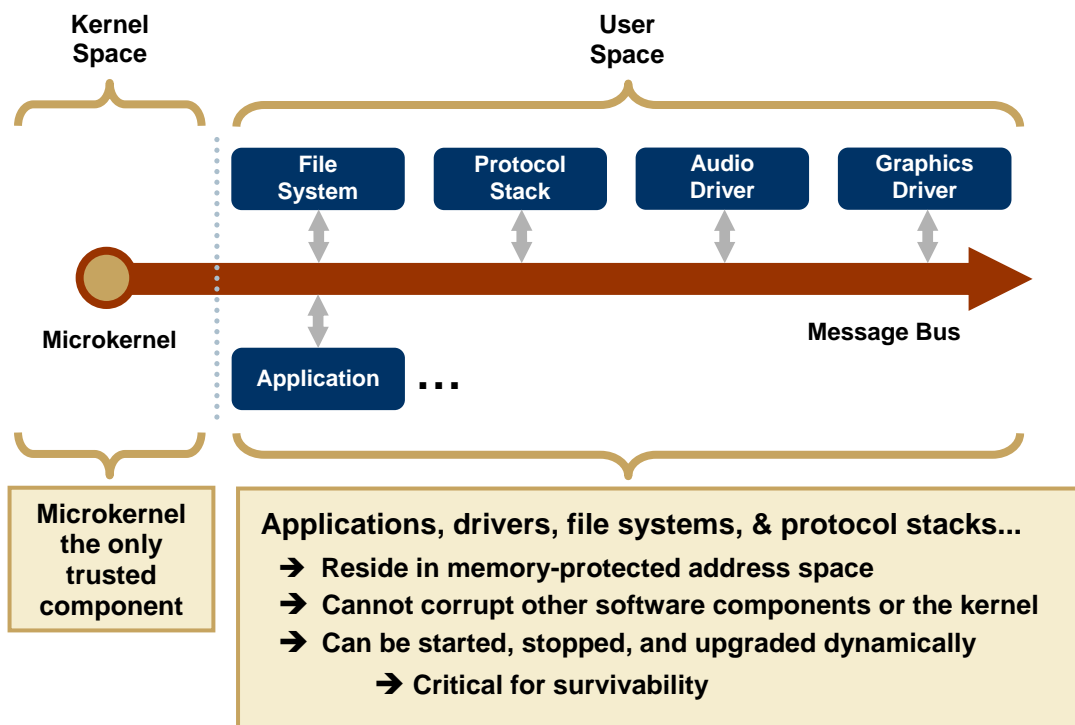


Figure 1 — QNX Neutrino microkernel architecture.

### 3.2 ANDERSON'S THREE PRINCIPLES

The basis for Anderson's *Computer Security Technology Planning Study* was to define how security operates within a computer product, regardless of type. His belief was that security could be distilled down to a few key principles and that these principles, if centrally controlled, would provide security functionality that could be verified by third parties. To accomplish this, he introduced the notion of the *Reference Monitor*.

The Reference Monitor is a simple concept that declares that every request for access to a given resource must go through the Reference Monitor. The Reference Monitor, which resides

within the kernel, cannot be circumvented, cannot be modified, and must be simple and compact enough to be readily understood.

Let's examine how these three key principles are assured in the implementation of the QNX Neutrino microkernel.

### *3.2.1 TAMPER RESISTANT*

The Reference Monitor must be inviolate when in operation. This requirement applies to all methods that can affect its operation, either programmatically or manually. If this integrity can be breached, either by malice, accident, or the system's inherent design, then the system becomes unusable with respect to user expectations and fails to behave correctly (i.e. as expected).

The QNX Neutrino RTOS implements its Reference Monitor within the kernel itself. The kernel and by extension the Reference Monitor is read-only and cannot be modified at run time because of its strict internal handling via the MMU. If the kernel image has been modified in an attempted attack or through an unintended action, it will not bootstrap into operational mode: the kernel does its own integrity checks at startup that ensure that it is sane and not damaged or modified. Since it cannot be modified while running, and will not start if modified, a running kernel can be trusted to carry out its operations correctly.

### *3.2.2 ALWAYS INVOKED*

It is essential that the Reference Monitor always be invoked, no matter what resource is being requested. The mechanism must be applied to all areas of resource allocation, regardless of the requester. The QNX Neutrino architecture is designed so that all resource requests are marshalled via the kernel: the Reference Monitor, through the application of the POSIX structure and strict use of the MMU, is always invoked and can be applied to all resource requests by the virtue of being situated inside the kernel.

### *3.2.3 SMALL AND SIMPLE*

The correct operation of the Reference Monitor is essential. It is a basic tenet that to be verifiable, a system design must be as simple as possible, while achieving all of its design goals. If the complexity of the implementation is too great, some holes might escape the notice of the designers and be exploited, intentionally or accidentally, by users. QNX Neutrino's design and implementation as a true microkernel operating system is simple, effective, and fully functional. The design is free of functionality that doesn't strictly meet the terse design goals and keeps only what is required for proper operation. Furthermore, QNX Software

System's engineering processes ensure that no part of the system falls through auditing cracks, from design to implementation to delivery.<sup>6</sup>

### 3.3 SALTER & SCHROEDER'S EIGHT PRINCIPLES

After Anderson's work on the Reference Monitor, Salter and Schroeder expanded upon the three principles with eight of their own.

#### 3.3.1 *ECONOMY OF MECHANISM*

Economy of mechanism is the notion of keeping it simple: the system does only what it's supposed to do. In other words, the system is implemented as efficiently and simply as possible and no simpler. Although this may seem to be an "obvious" requirement of security, all too many operating system kernels are overly complex and difficult to comprehend. Some kernels run into millions of source lines of code, all of which must be assured to operate correctly; otherwise, the security of the entire product is in question.

The underlying philosophy of the QNX Neutrino RTOS is that of modularity, simplicity, and compactness. The code base is designed to implement core functionality in as tight a package as possible, favoring conciseness. This approach helps ensure that the implementation is simple and straightforward, minimizing complexity so as to allow straightforward auditing during the implementation process. Because the QNX Neutrino microkernel is concise and modular, it becomes much easier for the QNX Neutrino development team to eliminate improper (and unknown) access paths.

The QNX Neutrino RTOS uses a well-understood and standard privilege mechanism: POSIX. This underlying structure and implementation allows QNX Neutrino to satisfy the concept of doing only what you need to do, without adding undue complexity that can harbor unknown holes and exploits. This industry standard is well-known and understood, and provides users with an explicit understanding as to how the system works, without any surprises. It is paramount that users know what to expect, and that the system deliver it, consistently. The QNX Neutrino RTOS achieves this via the POSIX compliance strategy.

This strategy satisfies the basic principles of a secure system.

---

<sup>6</sup> See the QNX white paper, "Engineering Confidence through Assurance Techniques."

### 3.3.2 FAIL-SAFE DEFAULTS

All too often, users have met with a computer failure only to realize that much of their work is gone or, worse yet, that the computer itself no longer functions. One of the tenets of security as defined by Salter and Schroeder is that a system is *fail-safe*. That implies that the system must have some set of *fail-safe defaults*. These defaults provide a well-known baseline against which the system can compare and determine how it should recover.

POSIX has an inherent concept of fail-safe defaults: explicit permissions must be set in order for components and processes to have access, the default privilege being to deny access.

Similarly, security-specific attributes of QNX Neutrino not only have known default settings but are set to a “known good default” during initialization. Hence, whenever new objects or subjects are created, their security attributes are assigned the “known good” initial default values. This ensures that no information is ever recycled or made available from previous users or processes that don’t belong to the owner of the new object. It also ensures that processes behave in a known fashion simply because all resources are set to known settings upon allocation.

In other words, object creation in the QNX Neutrino RTOS is specifically designed to ensure that every new object, be it a process, memory allocation, etc., is set to a known, neutral state, so that the unintentional transmission of information cannot occur.

Although this design ensures that any given process, including the overall QNX Neutrino system, begins from known good defaults, sometimes it is possible to roll back (or undo) the last operation or a series of operations, bounded by some limit, such as a period of time, thereby returning the system or a given process to a previous known “good” state. The QNX Neutrino RTOS was designed with this ability to roll back, or undo, the effects of an operation or series of operations while preserving system integrity.

### 3.3.3 COMPLETE MEDIATION

When someone usually thinks of “computer security” or something that is deemed “secure,” thoughts immediately turn to access control, or as it is known within the computer security arena, *complete mediation*.

Complete mediation is simply the notion that all access by a process to any given resource is mediated (i.e. permitted) by the kernel. This mediation is implemented within the kernel, ensuring that every access request goes through the same mechanism for verification, regardless of the nature of the request or which process is making the request. This mediation cannot be circumvented; furthermore, all access requests are evaluated consistently, and explicitly, to determine validity at run time with no special handling for *any* request.

More formally, access control consists of allowing or disallowing subjects (users, processes, nodes, etc.) from performing specific operations against objects (system resources controlled by the system). The decision to allow or disallow is based solely on the security attributes of the involved subjects and objects. As mentioned earlier, these security attributes are defined with known good defaults to ensure that the QNX Neutrino RTOS *always* starts in a known good state that is consistent between initializations.

Hence, within the QNX Neutrino RTOS, a typical access request would be defined as a rule:

*any subject with **owner** = admin  
may execute the read, write, create, and destroy operations  
on all objects of **type** = file.*

In this example, **owner** and **type** are security attributes.

Through its strategic use of the POSIX guidelines and system interlock hardware (via the MMU), the QNX Neutrino microkernel operating system ensures that the desired access controls described above are in place. The access rules integrated into the kernel cannot be bypassed, nor can they be modified in any way. This ensures that the system is always consistent and that user expectations are always satisfied with respect to using the system.

### 3.3.4 OPEN DESIGN

The QNX Neutrino RTOS adheres to the POSIX standard. It also heavily leverages the notions behind open design, ensuring that the application programming interfaces (APIs) are well-defined and behave in known ways. These APIs are documented thoroughly within the extensive QNX documentation set.

By leveraging known standards and publishing well-defined APIs, QNX Software Systems enables users, system builders, and system architects to *know* what the behavior of the system will be. The strategy of implementing the POSIX standard, for example, is a philosophy that is carried through on all aspects of the operating system; there is no false security through obscurity.

### 3.3.5 SEPARATION OF PRIVILEGE

The protection mechanism implemented by the QNX Neutrino RTOS allows for separation of duties (or *privilege*) where access to certain resources is required. This ensures a more resilient system and prevents accidental or intentional system disruption (see Table 1).

|                         | User thread<br>(uid !=0) | Root thread<br>(uid=0) | Root thread<br>with I/O privilege |
|-------------------------|--------------------------|------------------------|-----------------------------------|
| Maximum thread priority | User max (64 by default) | 255                    | 255                               |
| Map shared memory       | Based on permissions     | Yes                    | Yes                               |
| Map physical memory     | No                       | Yes                    | Yes                               |
| I/O operation           | No                       | No (x86)               | Yes                               |
| Attach to IRQ           | No                       | No                     | Yes                               |
| Disable interrupts      | No                       | No                     | Yes                               |

**Table 1** — Separation of privilege in the QNX Neutrino RTOS.

### 3.3.6 LEAST PRIVILEGE

Within QNX Neutrino-based systems, everything is based on providing *least privilege*. This approach allows the system and its components to operate with only the security privileges and system resources required to function correctly. This minimizing of unintentional or intentional damage to the system or parts thereof ensures that the system behaves in an understandable and consistent fashion and is completely deterministic. All privileges and resources must be requested from QNX Neutrino microkernel, which, prior to granting access, ensures that the requesting processes are permitted to request the given privilege or resource. The fine-grained partitioning provides the security required to limit side-effecting; it also ensures proper operation of the processes as well as of the underlying kernel infrastructure.

### 3.3.7 LEAST COMMON MECHANISM (RESOURCE PROTECTION)

A corollary to the principles of least privilege and separation of duties is the principle that resources are protected from implicit sharing. The QNX Neutrino RTOS leverages its inherent security mechanisms, not only to ensure proper allocation of resources, but also to ensure that the memory management unit protects resources and processes. QNX Neutrino thus provides strong assurance that no implicit and possibly unknown sharing or access to protected resources and processes will occur.

Similarly, when a process relinquishes control over specific resources, or over other processes, then those resources must be, and are, returned to the control of the QNX Neutrino microkernel. They remain inaccessible until the microkernel receives a request for the allocation of resources, at which time the microkernel automatically clears the resource to ensure that no information

leakage from previous uses is possible. Since resource allocation is implemented and controlled within the microkernel, which is known to be correct and unmodified, it guarantees that no covert channels or other means exist to allow access to residual information of protected resources.

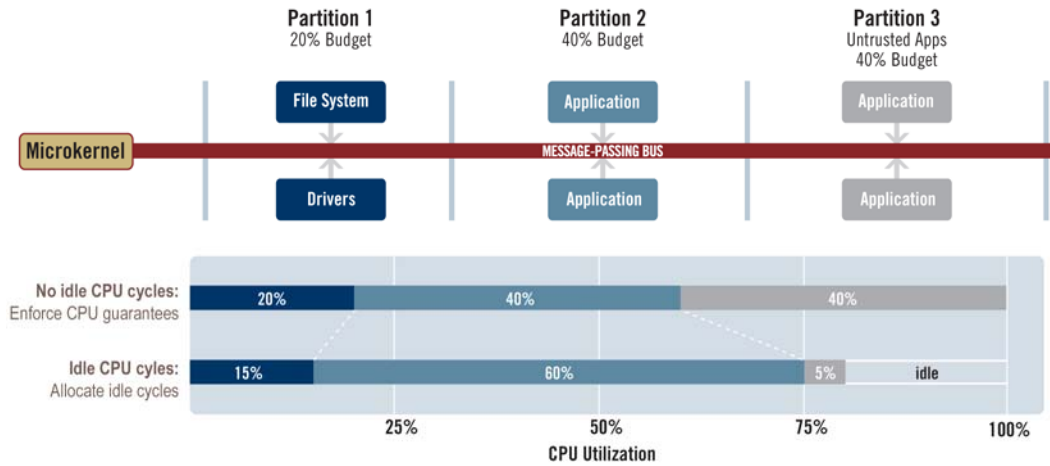
Furthermore, the QNX Neutrino RTOS gives users the option to request that their resources be cleared explicitly when releasing the resource: this provides individual threads with discrete control over their own information to make sure that no information is ever reclaimable, even within the same process. This approach effectively eliminates the possibility of unknowingly allowing retrieval of prior stored data within the same application. This is important, for example, when applications provide “plug-in” frameworks that allow third-party authored modules or tools to be dynamically inserted into an application authored by an originating party.

Hence, the QNX Neutrino RTOS provides the means to clear protected resources of any residual information from prior uses, either immediately upon reclamation by the microkernel or when the resource is reallocated for a subsequent use.

The QNX Neutrino RTOS further controls allocation of resources via *Adaptive Partitioning*. Above the regular scheduling of threads and processes, this technology can ensure that groups of threads or processes in a partition use only as much CPU time as is assigned to that partition. It also guarantees that partitions get the CPU time allotted to them. If any CPU time allotted to a partition goes unused, it isn't consumed via empty idling; rather, it is reallocated to other partitions dynamically.

This partitioning scheme ensures that a minimum of CPU time is given to every partition, as configured; it also prevents any process or thread from monopolizing all available overall CPU time. Properly planned partition schemes can also allow access to the system, regardless of how busy it is: you can set up a partition that is infrequently used (which would donate its budget to the rest of the system), but that would be available for diagnostics or other system maintenance.

All of the resource allocation and protection mechanisms within the QNX Neutrino RTOS are there to ensure resource availability and protection, thereby ensuring a more secure and *available* operating environment.



**Figure 2** — Adaptive partitioning enforces CPU budgets when the system is busy and allocates free CPU cycles during periods of lower processor utilization. In this scenario, Partition 2 consumes no more 40 percent of CPU cycles when the system is running at capacity. But it can consume more than 40 percent whenever other partitions require less than their allocated CPU budget.

### 3.3.8 PSYCHOLOGICAL ACCEPTABILITY

A system that behaves in a way that the user actually *expects* is said to exhibit *psychological acceptability*. In other words, there are no unexpected surprises. In more technical parlance, the system operates in a *deterministic* fashion. This means that the system operates as it should and in a well-documented manner. This is accomplished by having the QNX Neutrino RTOS use well-known and accepted standards as a basis for its security model. This model includes a definition of how QNX Neutrino behaves during initialization, self-test, recovery, and fault-tolerant operation.

## 4 MODERN ENHANCEMENTS TO SECURITY

During the intervening years since Anderson, Saltzer, and Schroeder presented their seminal papers on secure systems a few enhancements have been devised. Chief among these enhancements are accountability, application of priority to resources and processes, self-tests, and fault tolerance. QNX Software Systems has adopted these security elements into the very core of the QNX Neutrino RTOS to further enhance the security aspects of its microkernel architecture.

### 4.1 ACCOUNTABILITY

Today with Sarbanes-Oxley, privacy legislation, and other requirements for accountability, any system that manipulates information must meet at least rudimentary accountability requirements.

The original work in computer security considered accountability an afterthought. Today it is of paramount importance.

To ensure that the QNX Neutrino RTOS meets the emerging and evolving requirements for accountability and privacy, its audit facilities are defined on a strong, well-defined core.

Within QNX Neutrino, all events can be time stamped. These time stamps are generated via the RTC hardware and provide, in some instances, nanosecond resolution. The timestamps are provided by the kernel and aren't modifiable or configurable, ensuring that no clock skew exploits are possible with respect to audit subsystems and the like. This approach provides an added layer of security since system events that are recorded and audited cannot be repudiated with respect to the timing of their activities. The QNX Neutrino RTOS provides this crucial aspect; one that is found in hardened and high-end security systems.

The time stamp is utilized to ensure that each audit record is recorded consistently. Since the time stamp is controlled by the QNX Neutrino RTOS, the audit system — which leverages the built-in time stamp facility — is ensured that the time stamps associated with auditable events are correct.

## 4.2 PRIORITY OF SUBJECTS AND PRIORITY OF OPERATIONS

These elements are used to describe and verify how a system controls operations such that high-priority subjects or high-priority operations can proceed without undue interference or delay caused by low-priority subjects or operations. (By subjects, we refer to threads and processes carrying out operations on behalf of users, human or otherwise.) By its very nature and design, the QNX Neutrino RTOS incorporates a strict protocol to ensure that if resources become sparse (i.e. if there are more requests than available kernel threads), only higher-priority requests will float to the top of the processing queue.

System-level requests such as hardware interrupts are also subject to a strict prioritization protocol to ensure that high-priority requests never get “starved” out by lower-priority events. The QNX Neutrino RTOS guarantees that higher-priority requests are always processed first, making it extremely robust in high availability and mission-critical application environments, such as SCADA.

## 4.3 SELF-TESTS

As security researchers continued refining the fundamental requirements that defined *trusted* systems, self-tests were quickly included in the standard set of requirements. As every software vendor knows, it's crucial to perform at least a cursory test of a complex system upon startup to ensure its proper functioning.

According to these requirements, the system must test itself and verify the integrity of its stored executable code and objects. These requirements are needed to detect the corruption of the system or objects by failures that don't necessarily stop system operation. Such failures may occur either because of unforeseen failure modes or associated oversights in the design of hardware, firmware, or software, or because of malicious corruption of the system.

The QNX Neutrino RTOS was designed with uninterruptible operation in mind, and part of this design requires that the running kernel be in a perpetually known state, and by implication that it cannot be corruptible. QNX Neutrino incorporates built-in self-test mechanisms to ensure that it is always operating correctly and hasn't been modified. Upon startup, the kernel will verify itself to ensure that it is sane and unmodified. Correlating these tests with the knowledge that a running kernel is immutable ensures that once the kernel is operating, the user is assured of a correctly operating environment.

#### 4.4 FAULT TOLERANCE

Fault tolerance, together with the closely related requirement for fail-safe defaults, helps ensure that a system will maintain correct operation even in the event of a failure. If a failure occurs, the system must be able to properly recover to a "known good" state. The QNX Neutrino microkernel, by way of its self-tests, can, upon recovery, ensure that it has entered into a "known good" state.

Furthermore, should a process fail, its associated resources can be cleaned up and the process itself restarted from a "known good" state. If successive failures occur, the system can issue a notification to administrators, who can then take corrective actions.

### 5 CONCLUSION

As has been shown, the QNX Neutrino RTOS is based on strong security principles that are carried through to the very design of the underlying structures and procedures of the microkernel. The continuous efforts of QNX Software Systems to ensure that the QNX Neutrino RTOS provides a safe, reliable, and secure environment for customers' solutions represents one of the company's top objectives.

Anderson's three basic principles, coupled with Saltzer and Schroeder's eight defining principles of security design, provide a near-complete set of foundational elements that all good security-relevant products should attain. In fact, these principles are the fundamental components of good software engineering and, by extension, good kernel development.

Modern security experts equate the basic principles espoused by Anderson, Saltzer, and Schroeder with core security enforcement mechanisms necessary for any trustworthy product.

The role of the reference monitor, for example, is to validate all access attempts made to resources by any given process. As has been shown, this is exactly how the QNX Neutrino microkernel ensures that a resource is accessed, not only by the appropriate process but also by the right process operating against the correct data in the correct context.

In fact, the very definitions Saltzer and Schroeder described as defining security seem to have been tailor made for microkernels. Least privilege, economy of mechanism, mediation, open design, and separation of duties fairly defines what a microkernel *is*. And though the work done by Anderson, Saltzer, and Schroeder was, at the time, focused on what constituted a *secure system* it has become understood that what they've defined is also the definition of *good programming practices* and *good software design*.

The QNX Neutrino microkernel captures the necessary functionality required to meet the foundational definition of security as defined by Anderson, Saltzer, and Schroeder — a definition that holds to this day.

## ABOUT THE AUTHOR

*Eugen Bacic is Director, Emerging Technology at Bell Security Solutions Inc., and heads up security research and development. He has been a leading designer of information security technology for two decades, including research and development work on firewalls, public key cryptography, trusted audit, network and infrastructure security, malicious software detection, composable systems, policy engines, and security criteria.*

*Before joining Bell, Eugen leveraged his security research in founding Texar Corporation, a startup focused on information and policies. Prior to Texar he was Senior InfoSec Research Scientist at CSE. His many contributions while Senior InfoSec Research Scientist has been recognized by the IT security world and include author of the Canadian Criteria, the only non-US citizen on the congressional subcommittee tasked with developing the US Federal Criteria, the replacement to the US Orange Book, and the lead Canadian author of the Common Criteria. He has been a frequent member of security task forces in Canada and the United States.*