Softwarearchitekturen für ISO-26262-Systeme Teil 2

Als Reaktion auf die Nachfrage der Kunden nach mehr Anwendungen, Features und Diensten sowie aufgrund von Kostendruck konsolidieren viele Autohersteller nicht sicherheitsrelevante und sicherheitsrelevante Komponenten in ihren Fahrzeugen auf einer gemeinsamen Plattform. Welche Architektur – Microkernel-OS oder Hypervisor – das Mittel der Wahl ist, zeigt dieser Artikel.

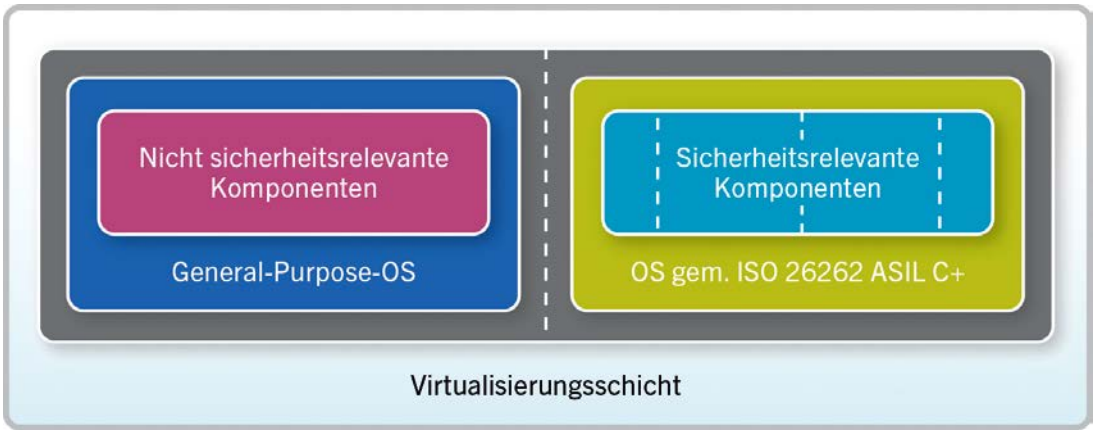

Bild 4: Auch mit Virtualisierung bleibt das Betriebssystem, auf dem die sicherheitsrelevanten Komponenten ausgeführt werden, dafür verantwortlich, diese Komponenten voneinander zu isolieren.

Ein Hypervisor kann dazu beitragen, die von ISO 26262 geforderte Komponentenisolation herzustellen: Auf der Virtualisierungsschicht könnten zwei Betriebssysteme ausgeführt werden, jedes in seiner eigenen Umgebung. Ein Betriebssystem betreibt die sicherheitsrelevanten Komponenten, während auf dem anderen Betriebssystem alles andere läuft, wie Multimedia-Anwendungen oder 3D-Navigation. Jedes Betriebssystem verhält sich so, als sei es das einzige Betriebssystem auf der Hardware, und nutzt die Ressourcen, die ihm die Virtualisierungsschicht zuteilt. Es gibt zwei grundlegende Virtualisierungstechniken:

- Bei einem Typ-1-Hypervisor laufen unterschiedliche Betriebssysteme auf der Virtualisierungsschicht.
- Ein Typ-2-Hypervisor führt ein Gast-Betriebssystem innerhalb eines anderen Betriebssystems (Host-Betriebssystem) aus.

Beurteilung einer Virtualisierungslösung

Die Virtualisierung besticht durch ihre scheinbare Einfachheit. Vor einer Entscheidung für eine Virtualisierungslösung sind jedoch zahlreiche technische und finanzielle Faktoren zu berücksichtigen. Ein großer Teil der Funktionalität der Virtualisierungsschicht ist hardwareabhängig. Nun ist die Hardware zur Unterstützung von Virtualisierung nicht komplexer als etwa eine Speicherverwaltungseinheit (MMU). Aber anders als die MMU-Technologie, die sich in jahrelangem Einsatz bewährt hat, ist die On-Chip-Unterstützung von Virtualisierung eine noch vergleichsweise junge Lösung. Beeinträchtigt ein Bug auf einem Chip die Verlässlichkeit oder die Isolation der Softwarekomponenten, muss entweder der Chip ersetzt oder ein Workaround für den Hypervisor und ggf. auch für das sicherheitsrelevante OS gefunden und implementiert werden – dies alles sind sehr teure Unterfangen.

Virtualisierung erweitert das System um eine weitere Softwareschicht. Neue Hardwaretechnologien haben zwar bereits viel dazu beigetragen, die durch die Virtualisierungsschicht entstehenden Latenzen auf ein Minimum zu reduzieren. Dennoch beeinflusst die Virtualisierungsschicht immer in gewissem Maße die Leistung der kritischen Komponenten. Besonders problematisch ist dies für Hardware-Peripheriegeräte, die eine hohe Bandbreite erfordern, wie etwa die

GPU. Bei einer Head-Unit, in der zum Beispiel ein Infotainmentsystem mit niedrigem ASIL und ein Fußgängerwarnsystem mit hohem ASIL integriert sind, müssen möglicherweise beide Systeme mit hoher Bandbreite auf die GPU zugreifen. Die Virtualisierungsschicht muss in diesem Falle die GPU-Ressourcen intelligent verteilen und dabei die Funktion des Fußgängerwarnsystems zu jedem Zeitpunkt garantieren können.

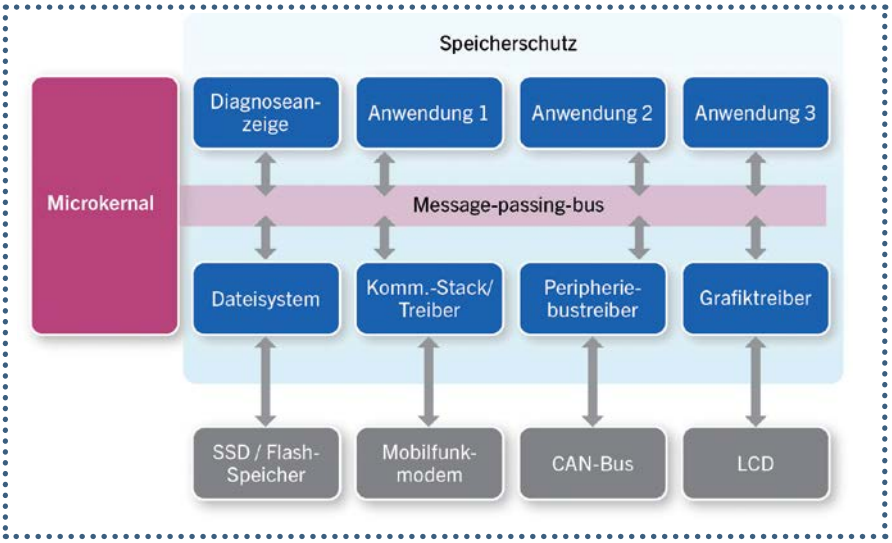

Bild 5: Ein Microkernel-OS isoliert die Komponenten gegeneinander, sodass sich ein Fehler in einer Komponente nicht durch das gesamte System ausbreiten kann.

Granularität

Virtualisierung isoliert zwar die beiden Betriebssysteme voneinander, aber sie isoliert nicht die Komponenten innerhalb ein und desselben Gast-Betriebssystems. Das Betriebssystem, auf dem die sicherheitsrelevanten Komponenten ausgeführt werden, muss diese zwar nun nicht mehr vor den nicht sicherheitsrelevanten Komponenten schützen, aber es muss weiterhin in der Lage sein, die sicherheitsrelevanten Komponenten untereinander zu isolieren.

Betrachtet man beispielsweise ein System, das aus einem Infotainment-System, einem digitalen Kombi-Instrument, einem Abstandsregeltempomaten und einem Spurhalteassistenzsystem besteht. Das Infotainment-System werde auf dem Gast-Betriebssystem A ausgeführt, während alle anderen Komponenten auf dem Gastsystem B laufen. In diesem Falle wären zwar die sicherheitsrelevanten Komponenten gegen das Infotainment-System isoliert, aber nichts schützt beispielsweise den Spurhalteassistenten vor Beeinflussungen durch das digitale Kombi-Instrument oder den Abstandsregeltempomaten. Ein solcher Schutz muss gemäß ISO 26262, Teil 6, Punkt 7.4.11, durch zusätzliche Isolations- und Abtrennungs-Mechanismen innerhalb des Betriebssystems B gewährleistet werden:

„Falls Software-Partitionierung ... benutzt wird, um Freiheit von Beeinflussungen zwischen Softwarekomponenten zu implementieren, muss gewährleistet sein, dass ... die gemeinsamen Ressourcen auf solche Weise benutzt werden, dass gewährleistet ist, dass die Software-Partitionen frei von Beeinflussungen sind." Bild 4 veranschaulicht, dass auch bei einer Virtualisierung das Betriebssystem B weiterhin die einzelnen sicherheitsrelevanten Komponenten mittels Partitionierung voreinander schützen muss.

	Design	Vorteile	Nachteile
Flache Architektur	Alle Komponenten werden gemeinsam in einem einzigen Speicheradressraum ausgeführt.	Effizient	Ein Zeigerfehler in einer Komponente kann Speicher korrumpieren, der vom Kernel oder einer anderen Komponente verwendet wird, und dadurch einen systemweiten Ausfall verursachen.
Monolithischer Kernel mit Speicherschutz	Die Anwendungen laufen als speichergeschützte Prozesse. Kernelkomponenten teilen sich ein und denselben Adressraum mit Dateisystemen, Protokollstacks und Treibern.	Der Kernel ist vor fehlerhaftem Anwendungscode geschützt.	Ein Fehler in einem Gerätetreiber oder einem anderen Dienst, der sich den Adressraum mit dem Kernel teilt, kann das gesamte System zum Absturz bringen.
Microkernel	Anwendungen, Gerätetreiber, Dateisysteme und Netzwerkstacks befinden sich in separaten Adressräumen, die gegen den Kernel sowie untereinander isoliert sind.	Fehler breiten sich nicht durch das ganze System aus. Das System kann eine ausgefallene Komponente neustarten. Komponenten mit unterschiedlichen ASILs lassen sich in einem System kombinieren.	Kleiner zusätzlicher Overhead für die Kommunikation zwischen den Komponenten.

Tabelle 2: Betriebssystemarchitekturen und ihre Strategien zur Komponentenisolation.

Kosten

Die Lizenzkosten für einen Hypervisor machen einen nicht unbedeutenden Teil der Stückkosten aus. Ein Design mit einem Hypervisor kann zwar die Hardwarekosten senken, reduziert jedoch aufgrund der Softwarekosten nicht notwendigerweise die Stückkosten insgesamt.

Hinzu kommt, dass die Verantwortung für ein sicherheitsrelevantes Fahrzeug-Softwaresystem nicht endet, wenn das Fahrzeug vom Band rollt, sondern sich über die gesamte Nutzungsdauer des Fahrzeugs erstreckt. Geht etwas schief, muss der Autohersteller das Problem beheben. Das heißt: Der Autohersteller oder seine Zulieferer müssen dazu bereit sein, die gesamte »

Alle Bilder: QNX

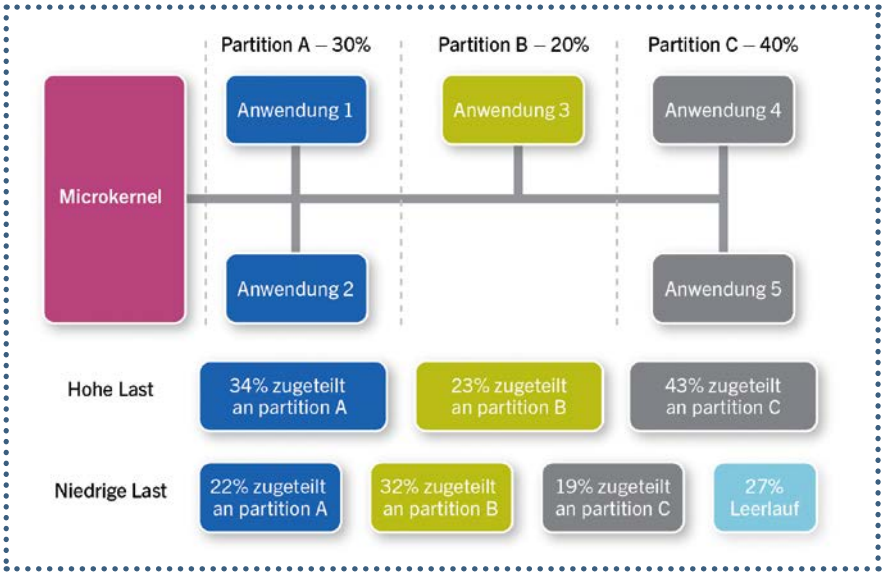

Bild 6: Ein Beispiel für adaptive Zeitpartitionierung.

Nutzungsdauer des Fahrzeugs über zwei getrennte Entwicklungsinfrastrukturen für zwei Betriebssysteme vorzuhalten, zu pflegen und die Kosten hierfür zu tragen.

Betriebssystemarchitekturen

Die Betriebssystemarchitektur ist bei einem ISO-26262-System aus zweierlei Gründen so wichtig: Erstens ist sie von grundlegender Bedeutung für die Verlässlichkeit des Systems insgesamt. Zweitens hängt von ihr ab, wie einfach (oder schwierig) und aufwendig es ist, Komponenten mit unterschiedlichen oder auch identischen ASIL-Anforderungen voneinander zu isolieren und zu schützen. Tabelle 2 enthält die häufigsten Betriebssystemarchitekturen für Embedded-Systeme und fasst zusammen, wie sich diese Architekturen jeweils auf die Isolation der Komponenten auswirken.

Ein Fahrzeugsystem enthält in den meisten Fällen eine Multimediakomponente, die High-End-3D-Grafik zur Anzeige unkritischer Informationen auf dem Bildschirm der Head-Unit nutzt. Diese Komponente benötigt ggf. nur einen ASIL von B oder sogar A, wohingegen die sicherheitsrelevanten Komponenten (für Bremsvorgänge, Abstandsregeltempomat, Parkassistenz usw.) nach ASIL C oder höher zertifiziert werden müssen. Ein Betriebssystem mit Microkernel bietet sowohl eine hinreichende Verlässlichkeit als auch ausreichenden Schutz vor Beeinflussungen für ein ISO-26262-System (Bild 5).

Schutz vor Beeinflussungen

Im Allgemeinen ist es bei einem System mit sicherheitsrelevanten Komponenten das Beste, mit einer Vielzahl komplementärer Techniken so viele Komponenten wie möglich voneinander zu isolieren. Diese Techniken betreffen unterschiedliche Stadien des Projekts, angefangen beim Design bis hin zur Validierung der fertiggestellten Komponenten und des Gesamtsystems. Die folgenden Betriebssystem-Features können wirksam dazu beitragen, die im Abschnitt „Beeinflussung“ angesprochenen Arten von Beeinflussung zu unterbinden.

Verhindern von Ressourcenblockaden

Mittels Ressourcenlimits (rlimit-Parametern) können Systementwickler Obergrenzen für Größe und Anzahl der Ressourcen (Adressraum, Arbeitsspeicher, Anzahl der Prozesse oder Threads, Anzahl der Dateideskriptoren usw.) setzen, die einem Prozess oder einer Anwendung zugeordnet werden können. Somit kann ein einzelner Prozess oder eine einzelne Anwendung nicht mehr alle Ressourcen an sich binden und daher auch keinen Ressourcenmangel bei anderen Prozessen auslösen.

Eine weitere mögliche Verteidigungslinie ist ein Programm zur Erkennung von Anomalien. Ein solches Programm lernt, welches Verhalten eines konkreten Systems als normal gilt, überwacht die Ressourcenzuteilungen und greift korrigierend ein, wenn es feststellt, dass ein Prozess eine ungewöhnliche Ressourcennutzung zeigt. Bound Multipressing (BMP) kann ebenfalls die Ressourcen der sicherheitsrelevanten Komponenten schützen. BMP ist eine weiterentwickelte Form von Prozessoraffinität bzw. SMP (symmetrische Multiprozessorarchitektur), bei der die Entwickler Threads oder ganze Threadhierarchien konkreten Prozessorkernen zuweisen können. So könnte bei einem ISO-26262-System auf einem Dual-Core-Prozessor Kern A nur Threads für sicherheitsrelevante Komponenten und Kern B nur Threads für alle nicht sicherheitsrelevanten Komponenten ausführen. Somit hätte eine auf Kern B laufende Multimediakomponente keine Möglichkeit, den sicherheitsrelevanten Prozessen auf Kern A Rechenzeit wegzunehmen.

Verhindern von Rechenzeitmangel

Zeitpartitionierung kann sicherstellen, dass alle Prozesse genügend CPU-Zyklen zur Erfüllung ihrer zeitlichen Vorgaben erhalten. Diese Technik teilt die CPU-Zeit in Partitionen auf »

und garantiert, dass jeder Prozess oder jede Gruppe von Prozessen einen vorgegebenen Anteil der CPU-Zyklen erhält. So kann kein Prozess mehr einen anderen Prozess aushungern.

Eine spezielle Form der Zeitpartitionierung ist die adaptive Zeitpartitionierung, die dieselbe Sicherheit bietet, aber auch dafür sorgt, dass keine Systemressourcen verschwendet werden. Bei diesem Verfahren wird einer Gruppe von Threads ein Mindestanteil an Prozessorzeit zugewiesen, allerdings nur dann, wenn die Threads diesen auch benötigen (siehe Bild 6 unten). Die vorgegebenen Partitionsgrenzen werden nur erzwungen, wenn die CPU mit voller Last läuft. Kann jedoch ein Prozess in einer Partition von weiteren CPU-Zyklen profitieren, während die Prozesse in anderen Partitionen die ihnen zugewiesene Rechenzeit nicht ausnutzen, kann das Betriebssystem die Grenzen anpassen und nicht benutzte CPU-Zyklen an Prozesse verleihen, die Verwendung für sie haben.

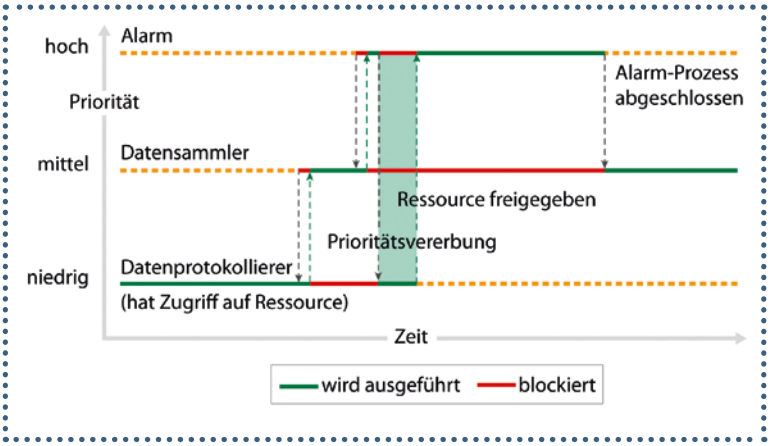

Bild 7: Prioritätsvererbung kann vermeiden, dass niederpriore Threads die Ausführung von Threads mit höherer Priorität blockieren.

Bei einfacheren Systemen können Scheduling-Verfahren wie RMS und DMS (Rate-Monotonic Scheduling und Deadline-Monotonic Scheduling) sicherstellen, dass Prozesse ihre Echtzeit-Deadlines einhalten. Bei RMS erhalten Prozesse eine umso höhere Priorität, je öfter sie ausgeführt werden. Für manche Systeme lässt sich dann mathematisch beweisen, dass die Echtzeit-Deadlines eingehalten werden.

Verhindern von unzulässigen Speicherzugriffen

Eine Hardware-MMU kann in Zusammenarbeit mit dem Betriebssystem illegale Speicherzugriffe vermeiden. Falls das OS diese Funktion unterstützt, hindert die MMU einen Prozess daran, den Speicher eines anderen Prozesses auszulesen oder zu beschreiben. Diese Funktion sollte jeder Softwarearchitektur abverlangt werden, die für ein sicherheitsrelevantes System in Betracht gezogen wird.

Verhindern von Datenfehlern

Techniken zum Schutz vor Datenfehlern sind etwa Prüfsummen, einfache Replikation, Datendiversifizierung sowie Plausibilitätsprüfungen. Unter diesen Optionen benötigen Prüfsummen bzw. CRC-Werte (Cyclic Redundancy Check) wohl am wenigstens Arbeitsspeicher, allerdings kann diese Technik fehlerhafte Daten nur erkennen, aber nicht reparieren. Bei der Replikation werden die Daten an mehr als einen Speicherort kopiert. Einer dieser Speicherorte kann sich auf einem Remote-System befinden. Bei der Diversifizierung werden dieselben Daten mit jeweils unterschiedlicher Semantik an mehr als einem Speicherort gespeichert. Plausibilitätsprüfungen stellen sicher, dass die eingelesenen Daten innerhalb akzeptabler Parameter liegen, und weisen anomale Daten zurück.

Alle diese Techniken erhöhen den Arbeitsspeicher- und/oder CPU-Overhead. Allerdings können sie bei Systemen, von denen garantierter Betrieb unter widrigen Bedingungen gefordert wird, äußerst nützlich sein. Deswegen sollten die Kosten für ihre Implementierung in der Designphase des Systems in Betracht gezogen werden.

Verhindern von Plappern

Plappern (Babbling) und andere böswillige Denial-of-Service-Attacken können durch ein Anomalie-Erkennungsprogramm eingedämmt werden. Die Anomalie-Erkennung lernt, was normales Verhalten ist, und greift korrigierend ein, wenn das System ein Verhalten außerhalb der erwarteten Parameter zeigt. Für diese Art des Testens ist auch das sog. „Fuzzing" nützlich. Hierbei werden Programmfunktionen absichtlich mit missgebildeten Eingabedaten oder auf unerwartete Weise aufgerufen, um Bedingungen aufzudecken, die das System ggf. nicht korrekt verarbeiten kann.

Verhindern von Deadlocks

Deadlocks treten auf, wenn zwei zusammenarbeitende Prozesse gegenseitig auf die Vollendung eines Arbeitsschritts des jeweils anderen Prozesses warten. Da jeder Prozess auf den jeweils anderen Prozess wartet, gibt es keinen Fortschritt mehr. Deadlocks können durch Prioritätsvererbung verhindert werden. Diese Technik löst das Problem der Prioritätsinversion, bei welcher ein Task mit niedrigerer Priorität verhindert, dass ein Task mit höherer Priorität seine Arbeit fertigstellen kann. Dazu wird die Priorität eines blockierten Threads mit hoher Priorität so lange dem niederprioren Thread zugewiesen, der die Blockade ausgelöst hat, bis dieser abgeschlossen ist (siehe Bild 7 unten).

Auch Hardware-Watchdogs können Deadlocks vermeiden helfen. Sie können jedoch nicht jeden Deadlock erkennen, denn ein Hardware-Watchdog ist blind für Deadlocks von Prozessen, von denen er gar nicht erwartet, dass sie ihn anstoßen. Im Gegensatz dazu kann ein Software-Watchdog oder ein Hochverfügbarkeitsmanager dazu beitragen, den Systemfortschritt sicherzustellen. Sofern die Systemarchitektur zulässt, dass einzelne Komponenten gestoppt und neugestartet werden, ohne dass das ganze System neugestartet werden muss, kann ein Software-Watchdog die von dem Problem betroffenen Prozesse neustarten, während der Rest des Systems seine Arbeit fortsetzt.

Zusammenfassung

Als Reaktion auf die Nachfrage der Kunden nach mehr Anwendungen, Features und Diensten sowie aufgrund von Kostendruck konsolidieren viele Autohersteller nicht sicherheitsrelevante und sicherheitsrelevante Komponenten in ihren Fahrzeugen auf einer gemeinsamen Plattform. Ein Betriebssystem mit Microkernel kann alle Betriebssystem-Features bieten, die notwendig sind, um sowohl der Nachfrage der Kunden zu entsprechen als auch sicherzustellen, dass das System seinen Sicherheitsanforderungen genügt. Bei einem Microkernel-OS ist der privilegierte Code sehr klein und weist einen gut getesteten und kurzen Ausführungspfad auf. Nur er erhält systemweite Privilegien. Kurz: Ein Betriebssystem mit Microkernel ist inhärent für sicherheitsrelevante Systeme geeignet.

Werden aus anderen Gründen zwei verschiedene Betriebssysteme auf ein und derselben Hardware benötigt, ist ein Hypervisor das Mittel der Wahl. Bei QNX ist man der Auffassung, dass bei der Entwicklung sowohl von Fahrzeug-Infotainmentsystemen als auch von sicherheitsrelevanten Systemen ein einziges Betriebssystem mit Microkernel alle benötigten Features bieten kann – sowohl für Infotainment als auch zur Erfüllung der von ISO 26262 vorgeschriebenen Garantien bezüglich Verlässlichkeit und Isolation. ■ (oe)

» www.qnx.de

Yi Zheng ist Product Manager bei QNX für Safety and Security.