Applying MILS principles to design connected embedded devices supporting the cloud, multi-tenancy and App Stores

Embedded, connected devices are nothing new. Developers have been establishing Machine to Machine communications for 20 or 30 years, long before the Internet of Things was ever conceived.

But until quite recently, embedded applications have tended to be static, fixed function, device specific implementations. In the current environment of ever quickening technological change, morphism and evolution are the order of the day. Now we see manufacturers and service providers all seeking to monitor, upgrade, enhance and supplement software implementation on a continuous basis.

In the enterprise world, cloud computing and “X”-as-a-service (PaaS, IaaS, SaaS) architectures have dramatically changed the economics of IT such that the infrastructure is now a utility or service rather than a capital investment. This has enabled significant innovations in service, including App Stores - libraries of products and services that are supported by the infrastructure and provide single instances of cloud-based software for the benefit of multiple “tenants”.

In the IoT, embedded devices are somehow connected to the “cloud”. However, the benefits of the IoT are unlikely to be realized unless the connected embedded devices are capable of supporting the multi-tenanted architectures that are common in enterprise IT. This implies that IoT devices need to provide secure separation between the different tenants accessing the IoT infrastructure.

For example, imagine a car as an IoT gateway device, with many demands on its systems infrastructure. Of course, the manufacturer will be first in the queue, providing software updates to security critical aspects of the vehicle and monitoring its condition. But alongside this core functionality there will also be a host of Apps from the Store to enable and entertain. The navigation service provider, accessing continuously evolving road data. Insurance applications, monitoring probationary drivers to help minimize premiums. Media streamers with the latest movies. Games, advertisements, and a host of other possibilities as yet unrealized.

Multi-tenancy without security is not enough. As if to highlight that point, at the 2010 IEEE Symposium of Security and Privacy, researchers from the University of Washington and the University of California – San Diego remotely accessed the systems on a late model car to unlock the doors, start and stop the engine, lock brakes and disable windscreen wipers – all potentially life threatening actions[1].

In their paper on the subject, the researchers observed that

“While the automotive industry has always considered safety a critical engineering concern (indeed, much of this new software has been introduced specifically to increase safety, e.g., Anti-lock Brake Systems) it is not clear whether vehicle manufacturers have anticipated in their designs the possibility of an adversary. Indeed, it seems likely that this increasing degree of computerized control also brings with it a corresponding array of potential threats. Compounding this issue, the attack surface for modern automobiles is growing swiftly as more sophisticated services and communications features are incorporated into vehicles.”

It is precisely these conflicting demands for security and accessibility which makes this and similar environments quite so challenging.

[1] Experimental Security Analysis of a Modern Automobile http://www.autosec.org/pubs/cars-oakland2010.pdf

Designing an IoT Gateway

Generalizing the automotive example to the wider Internet of Things, it is useful to consider exactly what properties are desirable in an IoT Gateway. Traditional embedded designs using COTS embedded operating systems can present many compromises.

Traditional embedded gateway designs rely on monolithic architectures, such that applications are hosted by a single operating system and all I/O support, management controls, and security controls are integrated into the operating system kernel. This monolithic construction creates the following challenges:

- **Fragile Single Point of Failure**
  IoT gateways are highly susceptible to attacks and failure due to their wide exposure to the internet across so many interfaces supporting a great deal of complex functionality. Because monolithic designs host all applications, perform I/O, and management functions, any failure in security policy or kernel coding flaw can jeopardize the security and availability of the system which will greatly limit the ability to operate in stringent safety markets.
- **Weak Separation**
  In order to provide a platform as a service a strong degree of separation will be needed to prevent collocated applications from compromising the privacy or availability of other applications. Relying on a monolithic operating system to provide separating computing environments is risky given the number of ways monolithic operating systems can fail or be subverted.
- **Limited I/O Support**
  With a monolithic design, all sensor and network interfaces drivers and I/O stack support must be built into the operating system kernel. If drivers do not exist for the selected OS, it can be difficult to support all varieties of desired sensor support and network interfaces.
- **Limited Application Support**
  Certain applications may only be available for certain operating systems. Choosing an OS that provides the best IO support, application support, ideal deterministic behaviour, and secure design may not be feasible.
- **Limited Online Maintenance**
  Due to highly complicated interdependencies of functionality in monolithic operating systems, the ability to patch or upgrade kernel functionality while maintaining platform operation is highly limited. The majority of kernel maintenance procedures require platform reboots which pose significant challenges for platforms that strive for full autonomy.

  The majority of issues in relying on a monolithic operating system as the foundation of an IoT gateway stem from limited application and IO support, and exposure to a single point of failure. A better approach is to have a more modular design that can provide more interoperability options and achieve higher levels of reliability and assurance.

Given these traditional designs are compromised, it is useful to consider relevant academic principles as a basis for a better solution.

The application of MILS principles

The solution to this conundrum lies in the MILS (Multiple Independent Levels of Security/Safety) initiative. MILS is a high-assurance security architecture based on the concepts of separation and controlled information flow; implemented by separation mechanisms that support both untrusted and trustworthy components; ensuring that the total security solution is non-bypassable, evaluatable, always invoked and tamperproof[2]. A MILS compliant distributed secure system for the connected car will therefore comprise of high-assurance components and applications which can be independently

[2] https://en.wikipedia.org/wiki/Multiple_Independent_Levels_of_Security

developed, modularly combined, evaluated and certified to adhere to these principles, in all aspects of its development.

This paper outlines the principles and application techniques surrounding three enabling technologies.

1. Least Privilege Separation Kernel Hypervisor,
2. MILS based network components for network traffic encryption and the consolidation of multiple encryption tunnels
3. MILS based data handling components to encrypt data and consolidate multiple storage partitions

Least Privilege Separation Kernel

The foundation of any such MILS compliant system is the Separation Kernel, a concept first mooted in 1981 by John Rushby[3]. A separation kernel consists of "a combination of hardware and software that permits multiple functions to be realized on a common set of physical resources without unwanted interference". By implication that suggests that the only interaction between the "security blocks" is by design, and that primary information flow will be from high to low security blocks (Figure 1).

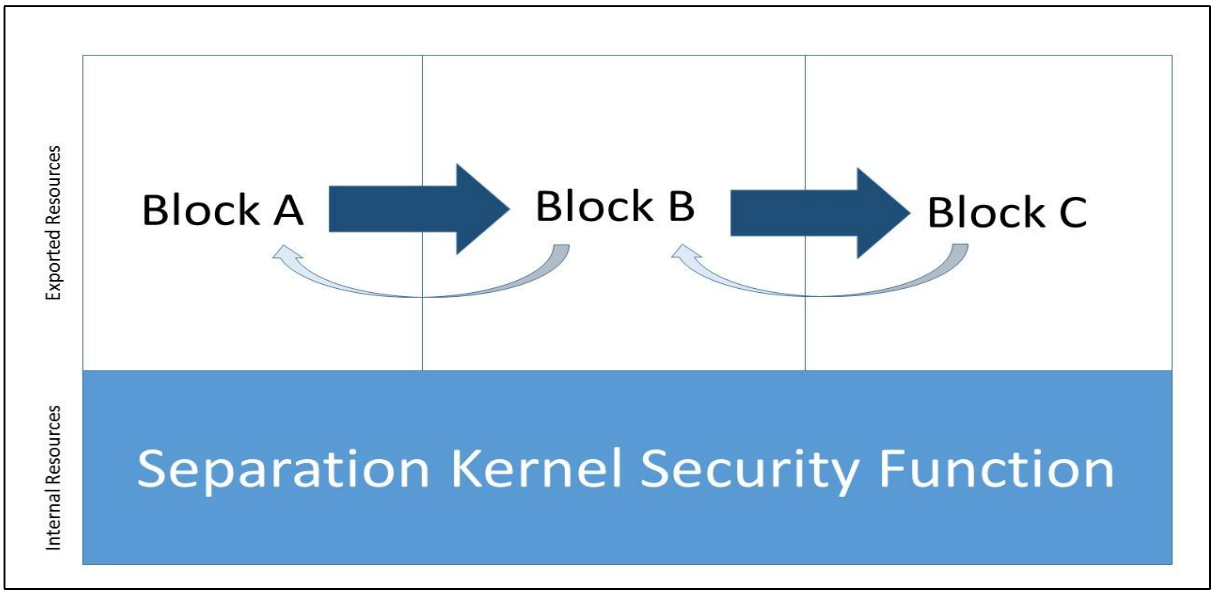

Figure 1. Primary information flow is from high to low security blocks – but some flow is required in the opposite direction.

A few years earlier in 1975, Saltzer and Schroeder[4] established a similar set of principles based on the idea of modularization, noting that "Every program and every user of the [operating] system should operate using the least set of privileges necessary to complete the job". The concept of a separation kernel clearly goes some way towards that goal, but because separation kernels are traditionally based on resource isolation there was insufficient granularity to take that principle to its conclusion.

[3] Design and Verification of Secure Systems. John Rushby, Computer Science Laboratory, SRI International. 1981.
[4] Saltzer and Schroeder, The Protection of Information in Computer Systems, ACM Symposium on Operating System Principles (October 1973)

That coarse granularity, coupled with an inevitable requirement for at least some flow of information from low to high security blocks, results in considerable emphasis being placed on developers if the boundaries are not to be breached.

The idea of marrying the two was therefore proposed by Levin, Irvine and Nguyen[5] resulting in the concept of a Least Privilege Separation Kernel, where both resources and subjects (executable entities) could be modularized. In this way, no subject needs to be given more access than required to allow the desired flows (Figure 2).

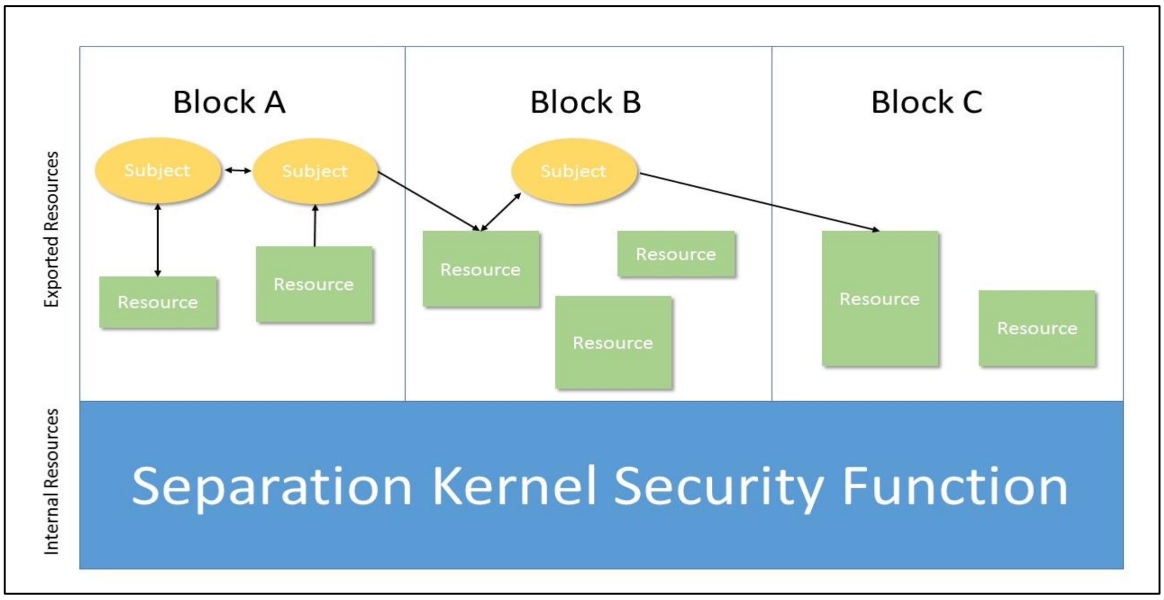

Figure 2. Superimposing Least Privilege principles on to those of a Separation Kernel combines per-subject and per-resource flow control granularity

In practical terms, the modularity of this approach enables the creation of de-privileged configuration utilities, device drivers, guest virtualization, device virtualization and management services. Hypervisor functionality is almost a by-product of this architecture, in that the subjects illustrated in Figure 2 can consist of anything from a minimally sized, "bare metal" application to a fully featured Real Time or General Purpose Operating System.

MILS based network and data storage components

In the context of our automotive example, this Least Privilege Separation Kernel Hypervisor may well be a well-founded principle, but it is only useful if it can be applied to provide the secure environment we need to ensure that the car as an "IoT gateway device" is not vulnerable to the attacks of the troublesome researchers and others with more malicious intent.

This has to be achieved, however, in the context of an existing infrastructure consisting very largely of the public internet. The onus for security therefore falls on the gateway devices – and in our case, that means the systems contained within the car.

There are also cost considerations to consider, particularly in the case of the ever expanding functionality of the modern car. To that end, it is important that the high build and maintenance costs associated with multiple vehicle networks should be avoided.

[5] T. E. Levin, C. E. Irvine, and T. D. Nguyen. Least privilege in separation kernels. In J. Filipe and M. S. Obaidat, editors, *E-business and Telecommunication Networks; Third International Conference, ICETE 2006*

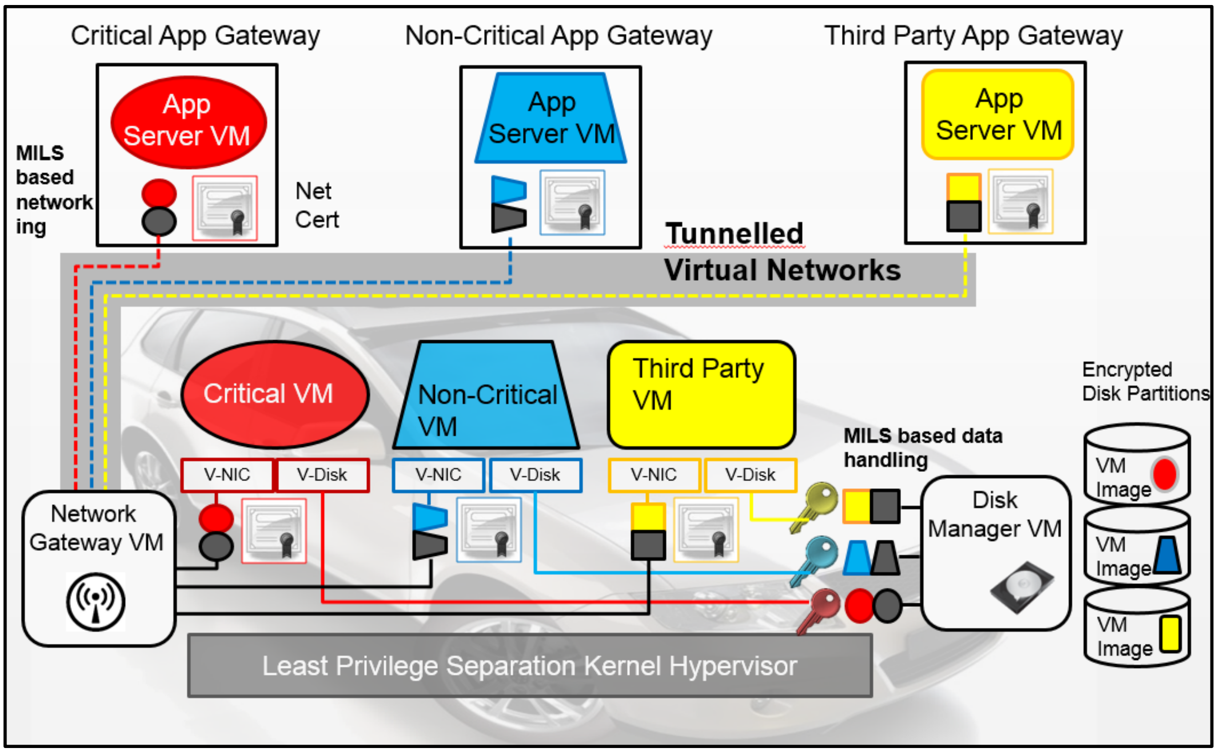

Figure 3. Superimposing Least Privileged subjects to handle data storage and networking functionality on the Least Privilege Separation Kernel Hypervisor.

Figure 3 shows how such a system could be designed in practice. The networking and data handling are implemented as minimalist least privilege “subjects”, tightly coded and running as “bare metal” applications to minimize their footprint and hence vulnerability.

The configuration of the system is completed statically at installation time, and the facilities for modifying that configuration do not exist once the system is running.

Three virtual machine (VM) subjects provide the means to run the Apps, possibly mapped to the individual cores of a multicore processor where there is hard real time dependency.

Encryption keys ensure the integrity of three tiers of data security, while the Least Privilege Separation Kernel provides the underpinnings to guarantee that the trusted code base underpinning the system is minimalized.

The net result is a robust solution, providing resilient application interfaces to prevent malicious software from subverting the virtual software architecture. It ensures the integrity of critical applications by protecting them from possible corruption from other application partitions. Costs are minimized and yet confidentiality is guaranteed through the application of a single network structure. And it can be ensured that vehicle applications are genuine, and that network encryption cannot be bypassed by server vehicle application VMs.

In summary, by applying the MILS principles throughout, the security/accessibility conundrum is resolved.

The Gateway based on the Least Privilege Separation Kernel Hypervisor

It is useful to summarize the pertinent capabilities and properties offered by Least Privilege Separation Kernel Hypervisors (more succinctly, “Separation Kernel Hypervisors”) in the context of the design of IoT gateways in general, and by implication, the connected car in particular.

- **Virtualization**
  Separation Kernel Hypervisors offer the ability to run multiple variants of guest operating systems concurrently on the same platform. This capability provides many options for hosting applications from multiple different operating systems and supporting a variety of different sensor devices and network interfaces. For example, through virtualization a design can be made where timing critical applications such as load-balancing and failover protocols run in real-time operating systems while running lessor critical applications in more feature rich operating systems. Virtualization is a key capability in dealing with interoperability challenges. Virtualization also plays a key role in supporting edge computing and platforms as a service, allowing analysis tools and tenants to use any operating systems and application best suited for the target task.
- **Bare-metal Applications**
  Some Separation Kernel Hypervisors provide the ability to run applications without the assistance of an operating system. Bare-metal applications are helpful for designs that require high performance or high assurance. Removing the dependency of an operating system reduces application overhead. For deployments that face stringent safety and security requirements, code reviews of critical applications may be necessary. Bare-metal applications can greatly aid the code analysis process by removing the complex code dependencies on the operating system.
- **Full Mediation**
  Separation Kernel Hypervisors provides the ability to explicitly monitor and control all platform transactions. This allows system architects to gain full awareness of the state of the system that can be used to help improve platform availability and assurance.
- **Isolation**
  Separation Kernel Hypervisors partition hardware resources and assign them to either Guest OSs or bare-metal applications according to a specification defined by a system architect. Separation Kernel Hypervisors provide the strongest degree of protection of resources using the native capabilities of the CPU. Strong resource isolation is a key enabler for achieving high availability.
- **Robustness**
  Because Separation Kernel Hypervisors serve as the central host of the computing platform, they are designed specifically to be highly reliable and resilient to malicious techniques of subversion. Separation Kernel Hypervisors achieve this property by relying on simple least privilege internal designs with limited functionality and limited exposure to hosted software. Separation Kernel Hypervisors tend to be ten to hundreds of times smaller than monolithic operating systems.
- **Determinism**
  Separation Kernel Hypervisors provide the ability to control explicit execution schedules for guest operating systems and bare-metal applications. Coupled with the ability to isolate resources for software modules, Separation Kernel Hypervisors can guarantee the availability of a system, ensuring any critical application can never be pre-empted or starved by competing applications. The deterministic control Separation Kernel Hypervisors have over a computing platform is a key enabler for designing highly reliable systems.
- **Certification**
  Separation Kernel Hypervisors are designed to support environments that face both security and safety regulation. Relying on the intrinsic properties and core capabilities of a Separation Kernel Hypervisor system architects are given the tools needed to design and implement

solutions greatly exceed the levels of reliability compared to a solution running on a monolithic operating system.

Least Privilege Separation Kernel Hypervisor under attack

All of these features clearly have merit, but ultimately the Separation Kernel Hypervisor itself is the critical component of the architecture; the trusted code base that makes the rest of the architecture possible. That make it a primary target for attack.

It would be bold or foolish to claim that it could never be compromised, but the theoretical principles which guided its design also highlight why that threat is optimally defended.

All attacks require an attack vector; a means of entry, commonly a NIC or device driver. LynxSecure offers no such direct attack vector because in accordance with the principles of a least privilege separation kernel, device drivers and communication mechanisms such as NICs are all placed within VMs.

The possible configuration permutations of LynxSecure are almost countless. The superimposition of Least Privilege principles on Separation Kernel blocks results in fine per-subject and per-resource flow-control granularity, and so communication paths only exist if the system architect wants them.

Where they must exist, should there be a concern that communication between VMs by means of a virtual Ethernet and its well-known protocol, then the architect might prefer to take advantages of the proprietary “hypercalls” API which adds an element of “security by obscurity” to the design.

Figure 3 shows an example of a communications mechanism between VMs by means of virtual network cards. That clearly involves communicating through LynxSecure, but not with it. In other words, for this to represent a vulnerability there would have to be coding deficiencies to compromise the integrity – and the very fact that it is so small and has been certified to very high standards means that the risk is tiny.

Finally, LynxSecure is statically configured. The mechanism required to modify that configuration does not exist at run time, and so there is no possibility of the installed configuration being modified by an attacker.

In summary; there has to be a communications path through the gateway for it to fulfil its purpose, and that implies the existence of possible means of compromise. But here, the risks are kept to an absolute minimum in accordance with the principles on which its design is founded.

Least Privilege Separation Kernel Hypervisor in practice

The example of an automotive application is highly topical (figure 4).

ANDY GREENBERG SECURITY 07.21.15 6:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

Applying MILS principles to design connected embedded devices supporting the cloud, multi-tenancy and App Stores

Figure 4. The vulnerability of the connected car is a highly topical subject, and a thorny one for the automotive industry

It is perhaps no surprise that it is presently under review by a number of vehicle manufacturers and their tier one suppliers, and that at the time of writing the commercial relationships involved mean that detailing their practical application here is not possible.

The security issue highlighted is new to the automotive sector, because until recently cars have been secure simply by virtue of being isolated. However, that is not true of all other fields of endeavour, and there are practical applications of this technology dating back for almost a decade. This state of affairs therefore has the potential to offer the automotive and other sectors a proven and seasoned solution to a new and challenging problem.

One field where security has always been paramount in in the military, and it is perhaps natural that Separation Kernel Hypervisors have therefore found many of their earliest practical applications in that field. Such applications clearly demand certification in accordance with military security standards. For example, a US armed forces program has used a Separation Kernel Hypervisor as the security foundation for an embedded mission critical security solution. This solution successfully underwent a security certification involving extensive penetration and other testing for use in DOD High Threat Environments, satisfying the following security requirements:

▪ Director of Central Intelligence Directive (DCID) 6/3 Protection Level (PL) 4 – separation of Secret to Secret/compartmented security domains in the presence of untrusted users.

▪ 8500.2 Mission Assurance Category I (MAC I) for systems handling information that is determined to be vital to the operational readiness or mission effectiveness of deployed and contingency forces in terms of both information system content integrity and timeliness/availability.

Although further details about this certification and others like it are classified, the artefacts created to achieve that certification are mostly generic and hence relevant to other certification processes.

Other "real life" applications include

- A UAV ground controller, providing a user interface and control platform for controlling unmanned vehicles. The use of a Separation Kernel Hypervisor in this environment provided
    - A path to certification,
    - Safety critical partitioning of the user interface from the control function,
    - Deterministic control, and
    - Flexible application options.
- An electronic "flight bag", consisting of a cockpit user interface for features including a map display an electronic forms. In this case, the Separation Kernel Hypervisor provided
    - The separation of a low integrity user interface from a high integrity aircraft bus
    - The optimal OS for each application, permitting the use of state-of-the-art graphics
    - A certifiable approach to isolating fully virtualised OS

Separation Kernel Hypervisor Based Design Strategies – an example

Suppose that there is a requirement for an IoT gateway in a traffic control application. In its simplest form, the gateway in the system is providing local control for a set of traffic lights. Road sensors provide information about what is happening to the traffic flow, the gateway provides local control to action the immediate policies, and the policy decisions are derived from the command centre based on the information fed back to it from the gateway.

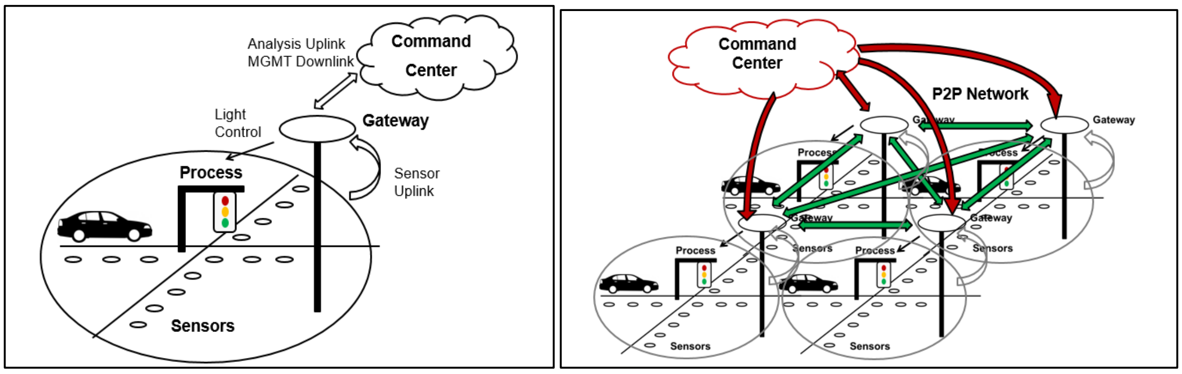

Figure 5. A simple IoT Gateway application providing local control for a set of traffic lights (left). More complex traffic control requires local peer-to-peer communication as well as policy decisions downloaded from a command centre (right)

As this system is expanded, so the intercommunication between gateways and the command centre becomes more complex. Immediate traffic flow issues are management by means of local P2P networking while the policy decisions are still downloaded from the command centre (figure 5)

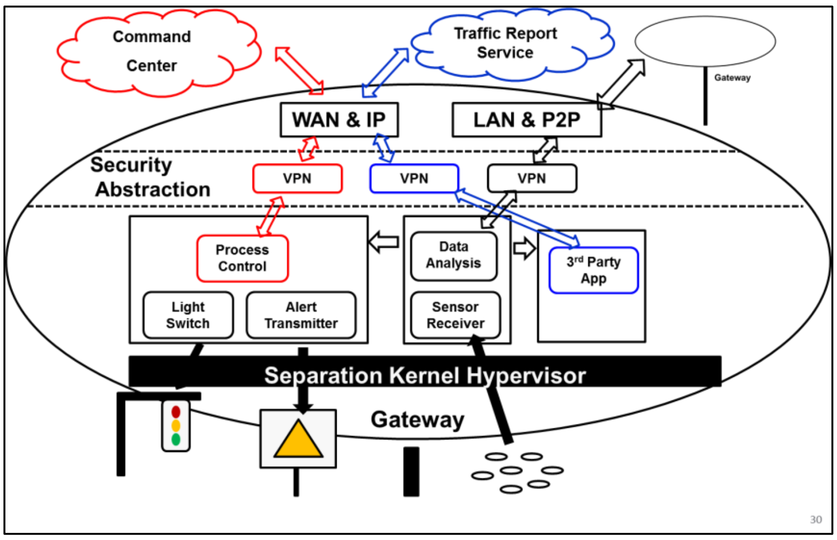

Figure 6. Ultimately the system requires multiple domains to permit the various tenants safe access to the same TCP/IP stack

Ultimately, the system requires third party access to collate data for traffic reporting; naturally a less critical domain requiring separation from the traffic control functionality of the system (figure 7).

In the context of this traffic control system, several attributes of the Separation Kernel Hypervisor based gateway are significant.

- Modular Composition

  Using the virtualization and bare-metal application capabilities, and relying on the isolation and deterministic properties of a Separation Kernel Hypervisor, platforms can be constructed out of a variety of software modules where individual functions of the platform can run in independent partitions and simple interfaces between modules can be defined so that the state of the system is well defined and understood.

Modular designs promote better options for interoperability and sustainability. For example, gateway maintenance routines such as new sensor device or network interface upgrades or bug patches can be performed without powering down the device or disrupting vital operational components.

In the traffic control system (figure 7), we see a partition hosting a VM dedicated to the local lights and signage, a second to the receipt and analysis of data, and a third transmitting data to a traffic report service. An additional VM running a bare metal application ("Security abstraction") takes care of the encrypted data from the different tenants.

- **Heterogeneous Computing**

  Relying on virtualization, architects can mix and match combinations of operating systems concurrently running on the same platform to host a wide variety of applications and I/O interfaces that may be exclusive to certain operating systems. Heterogeneous computing is an excellent strategy for supporting the most interoperable designs, and defending against proprietary vendor lock-in or obsolescence.

  The traffic control example might deploy a bare metal application for the Network Gateway VM and the traffic signal control, perhaps an RTOS for the data analysis, and a general propose OS for the traffic report data handling.

- **Fault Tolerance**

  Relying on the robust and highly available backbone of the Separation Kernel Hypervisor and modular design principles, system architects can create fault-tolerant designs using heart-beat and health inspection techniques. Some Separation Kernel Hypervisors allow guest operating systems to report their health status, policies can be set in a Separation Kernel Hypervisor to reboot or repair modules in the event of failed health status reports. Furthermore specialized custom modules can be created to inspect the health of other modules either through communication protocols or simply giving inspection modules access to view operating system and bare-metal application memory resources.

- **Security Abstraction**

  A good architectural approach is to design systems such that applications run in separated environments from security sensitive applications to prevent security controls from being bypassed or infiltrated. In the traffic control system, the "Security Abstraction" layer takes the form of a VM running a bare metal application takes care of the encrypted data from the different tenants to prevent users from disabling or bypassing the crypto function.

Conclusions

Monolithic architectures pose significant challenges in the ability to support stringent safety standards and meeting demanding market needs. With the use of a Separation Kernel Hypervisor and modular design techniques, IoT gateway vendors are given many options for building highly interoperable, reliable, secure, and sustainable solutions using low cost COTS components.

The automotive industry in particular has suffered by moving from a world where system isolation offers the ultimate security protection, to the connected car where that past assumption makes it especially vulnerable.

Such industries cannot discard all existing code and applications, and re-write them overnight. Separation Kernel Hypervisors offers the proven, certified technology to provide separation of that vulnerable software from the dangers of external public access.

This is not new technology. It is proven and has been certified and in use in the US military for almost a decade. It is not an operating system; not even a cut down operating system. It is a separation kernel, leveraging hardware virtualization to minimize both the trusted code base and the attack surface.