Real-Time Operating System Security

Weider D. Yu (Dipti Baheti, Jeremy Wai)

Computer Engineering Department, San Jose State University, San Jose, CA 95192-0180

Abstract — **Real Time Operating Systems lie at the heart of most embedded systems. Connectivity of these systems enable user to monitor and control these systems remotely. This report show current systems have many faults in which the system is connected to each other, and to the world. An analysis shows these systems could be significantly more secure if security policies were followed and if current computer networking security techniques are applied.**

Index

0.0 OUR CONTRIBUTIONS

We contributed to Sections:
5.1 Enforcement of security policies
5.2 Improving Existing SCADA
5.3 Dynamic Security policies through Heuristics
8.0 Lessons Learned

1.0 INTRODUCTION

The Real Time Operating System (RTOS) is a computing environment used in systems that require a response within very specific time constraints. The correctness of the system depends not just on the correct logical result but also on the delivery time of the result. For most applications, real time performance is the main criterion in evaluation of RTOS. RTOS employ several methods to ensure that it meets the real time system

requirements. This makes RTOS mostly a single purpose system.

There are two folds to security relating to RTOS. Firstly, security within the RTOS revolves around the ability to keep jobs separate so one task does not interrupt another. This problem becomes more complex as RTOS is needed for more complex applications while still keep its deadline requirement. Security of a RTOS also relates to the stability of the overall system because insecure tasks can cause the system to fall into an unknown state and crash. This is unwanted because RTOS are often used as mission and safety critical components of certain applications.

The second security issue has to do with RTOS in a network. RTOS seldom operate alone in modern day industry. It usually controls certain automated machinery and are often linked to a monitoring system that connects to other RTOS for management of an entire facility. It is used in wide variety industry include electrical power grids, traffic signals, water management systems, oil refineries, chemical plants, and pharmaceutical manufacturing. These control networks are often called *supervisory control and data acquisition* (SCADA) or *distributed control system* (DCS). These fields are merging as technology advances. They connect to their companies' database and mainframes running other operations, which are also connected to the internet. This meant that RTOS can potentially be reprogrammed by intruders to either crash or do harm to those RTOS that controls physical machinery. The obscurity of SCADA in the public eye gives SCADA the illusion of anonymity, but this is not security.

Security of both intra-RTOS communication and SCADA are similar to that of non real time operating systems and computer networks. As technology of control systems merge to that of computer networks, problems that control system encounters will mirror that of the computer network counterpart. This is a blessing and a curse because problems that plague computer networks will also plague SCADA, and this may have an even a bigger impact to the industry as SCADA networks requires high rate of reliability and RTOS often controls sensitive equipments that cannot malfunction.

2.0 DEFINITIONS

RTOS – Real-Time Operating System, real time in the sense that a process running will finish within the deadline specified, or else even if the process returns useful function, it would be too late. An example would be the antilock brake system in a car. If it returns wheel slip information too late, it would be useless.

Embedded Systems – Usually has a microcontroller at the heart of the system. It can vary in function depending on its programming. But for this article, it is the physical component that a RTOS sits.

Deadlines – In an RTOS, deadlines are the amount of time that the system can perform a process and return the results. If it cannot return the results before the deadline, the results might as well be incorrect.

SCADA – Supervisory Control and Data Acquisition, it is a network of machine's control units tied together to a workstation. The workstation is able to read relevant machine data. Workstation can optionally send commands to the machines. Machines could be anything from sensors, to motors opening and closing valves.

DCS – Distributed Control System, a network of controllers where each one controls one or more machinery. A system of controllers forms a network for monitoring and changes. Availability of off-the-shelf parts has made SCADA and DCS virtually the same.

Virus – A computer virus is a small software program that spreads from one computer to another computer and that interferes with

the computer operation. A computer virus may corrupt or delete data on a computer, use an e-mail program to spread the virus to other computers, or even delete everything on the hard disk.

Denial-of-Service attack – Denial of Service (DoS) is an attempt to make a computer resource unavailable to its intended users. A common method is to flood the target machine with false requests so that it cannot respond to legitimate traffic. Usually a DoS attack does not intend information theft or other security losses, but it can cost the target a great deal of time or money.

Encryption – A way to hide plain data with keys or passwords to deny unauthorized access into sensitive data.

VLSI – Very Large Scale Integration. It is the process of creating integrated circuits on a single chip by combination of very large number of transistor based circuits.

3.0 Related Technologies and Work

With advances in VLSI technology, embedded systems have become very inexpensive. Complex circuits can be achieved on chips of very small sizes. Therefore, embedded systems can be found in devices ranging from aircraft and military systems to industrial equipment, automobiles, personal devices and even smart toys. A number of embedded system applications require them to be connected to a network of some sort. This enhances an embedded system's utility and capability by enabling them to be remotely controlled and by allowing certain systems to download and implement new features and updates on the fly. Control systems installed in industries can use networks of embedded control nodes for various applications such as chemical processing, electrical power distribution and factory automation. The entertainment devices popular in homes are connected to the internet. The set-top boxes and gaming consoles can download new games and features on demand. Household appliances can be connected to a network to automate management of lighting, heating and also security.

One of the first examples of a functional Real Time Operating System implemented on a large scale was the Transaction Processing Facility developed by IBM. Since then, Real Time Systems have evolved and undergone many changes. Now there are several hundreds of Real Time OSs available. Some of these are very powerful and can be used for several general purpose applications while there are some simple implementations that are designed for only one particular purpose. Irrespective of the design of the Real Time System, its consistent ability to accept and complete a task within a limited time remains its key characteristic feature.

The connection of embedded systems to networks means that they are not only susceptible to security issues caused by their own design, but also to security issues faced as a result of interfacing to external networks. The widespread use of RTOS in various domains makes security a critical aspect in their design. Security issues within a RTOS system pose a problem that could slow the performance of the system. This is why most RTOS systems are especially configured to handle a certain job. This way, the whole system is optimized to do one and only one task, and thus would excel at it.

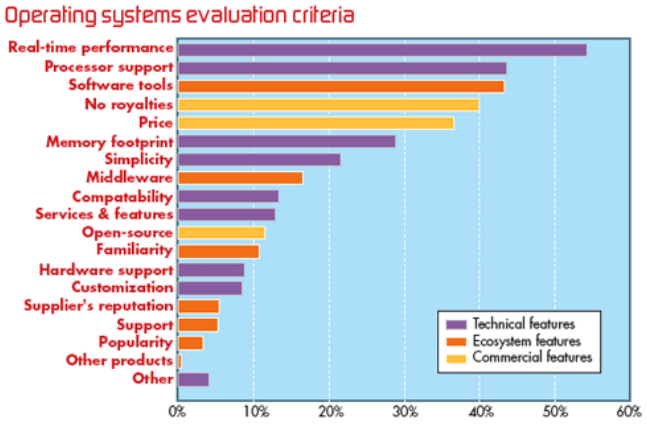

Figure 1. Relative importance of different criteria in selection of Real time operating system. (Ref: www.embedded.com)

As technologies advances and RTOS becomes easier to deploy (by being in an embedded system) and more practical than existing systems. More and more industry started utilizing RTOS to do a job that used to be done by people or simpler systems. Embedded systems with RTOS have become very common in large number of industries.

4.0. CURRENT SECURITY ISSUES IN RTOS

There are many issues surrounding the emergence of computer systems in the world of process controls and machine automation that used proprietary system. Such systems were often simpler and made from the ground up for the company's clients. But as the proliferation of small off-the-shelf devices made them easy to obtain, clients stopped seeing the use of companies that specialized in expensive process controls. Computer networking spread at the same time and the same industry saw the convenience of linking multiple embedded systems together to a monitoring station that enabled workers to control machinery more effectively.

Unfortunately, the connection of the embedded system to a network via which it receives instructions, control parameters and new programs also makes it vulnerable to attack from malicious sources. It could be done by an external party or even an internal miscreant. Since it is not necessary to change a program to make it misbehave, it can be done relatively easily by entering an invalid command or wrong parameters. Attacks on website such as Google, Amazon etc that result in denial of service, are not uncommon these days. These kinds of attacks can be replicated on embedded systems also. There have been incidents where hackers have achieved access to SCADA systems, thus compromising their security. In 1998, there was an incident where a 12 year old hacker got access into the computer systems of the Roosevelt Dam and gained control of the floodgate operation. Such attacks not only cause physical and economic loss, but can also endanger public safety. [9]

Often during the design of embedded system, the goal is to provide good performance and in the process of trying to maximize performance, the security aspect is not given due importance in the design requirements. The Operating System has the ability to control the memory and processor resources of a system. If the Operating System is not built with security in mind, it will always be vulnerable to attack and fail to prevent and limit the damage due to unauthorized access. Resistance to such attacks can be ensured only if security is built into the system architecture and implementation.

Once attackers gain unauthorized access to an embedded system, they have the potential to bring down the system in many different ways. Depending on the system that has been targeted, these attacks can cause great financial losses or even loss of life. The attacks can compromise the system's integrity, confidentiality, authenticity or availability. Sniffing the data transmitted across the network is an example of an attacker trying to gain access to confidential information. If proper encryption methods are not employed, once the attacker intrudes the network, he/she might be able to sniff the communications. An attacker could learn all the data and control commands while listening to the traffic and could use these commands later to send false messages. An attacker can also tamper with the data transmitted over the network and thereby compromise its integrity. The stored data can also be made target. This can cause devices to fail at a very low threshold value or an alarm to not go off when it should. Another possibility is that the attacker, after gaining unauthenticated access, could change the operator display values so that when an alarm actually goes off, the human operator is unaware of it. This could delay the human response to an emergency which might adversely affect the system's safety, the mission's success or may even jeopardize people's lives.[8]

4.1 INSECURE SCADA

SCADA is everywhere today. People do not see it because it is transparent or pervasive to their daily lives. It is usually monitored by some company or by government workers far away that people do not see. These include:

- Electric Power that goes to your house
- Water pumps that pumps water to your house
- What that goes out of your home
- Traffic lights
- Public transportation
- Air conditioning in the building you work in
- The cell phone you use
- Natural Gas that goes to your house

Most people that know about these systems either work in the industry or monitor the systems. So when these systems get attacked by intruders, people do not understand the type of impact it has, they only read it in the news once in a while and forget about it. A presentation [5] made by veteran security experts claim that many SCADA all over the country are vulnerable. There was often no authentication in the SCADA system. Old SCADA systems were never updated, only replaced after 5 years. New systems were patched every month. Companies running SCADA denied it was connected to the internet while it actually was connected to the internet. SCADA not made to connect to the internet were vulnerable if they were connected. SCADA intrusion by disgruntle employees were easy; it was made worse when they were connected to the internet. Penetrative tests, to see how vulnerable SCADAs are, are available widely. It was used to conduct tests by the researchers. Administrator and managers in various companies often lied or do not know about security vulnerabilities their system had. Wireless access points were proven to be a big problem. If IT in a company did not provide adequate access to the company's network, employees often connected their own wireless router to the network with no password protection. Some companies had multiple networks, one for labs and another for office; they did not realize that both networks were often connected together at multiple points. Companies with IT department often did not realize their network diagram did not match the actual configuration of their networks. Employees were allowed to connect their laptops either physically or through USB drives to the SCADA network, spreading viruses. Some SCADA systems involved in production of gas and oil had no protection at all. Large companies with websites had no network separation between their private network and their website. Penetrative test was able to go through the website to the company's private server using unpatched network. Private networks were simply setup with default passwords, because the customer never bothered to change it.

Even though these vulnerable SCADA systems are not well known, they could be found easily by looking on the internet. SCADA venders often advertize their clients to claim importance and at the same time provide detailed information from their site, from there it provides a vector into those SCADA systems. SCADA that are electronically secure would sometimes be physically insecure. Researchers found power substations that were unlocked with a computer running connected to all the equipment and the SCADA system. Some isolated workstations even have insecure wireless connection to SCADA. Dams also had insecure SCADA system where an intruder could actually release water from the dam to flood inhabited areas. Intruder could also disrupt other systems in the Dam. Vulnerability did not stop there. Even if tampering was detected, there was often no way to track down where it came from. There were no user authentication in the system, everyone used the same account. Tracking systems were often turned off or never enabled. Many of these intrusion tests done by the researchers were not difficult to perform.

The researchers found that many security policies were simply not enforced; employees often did not follow protocol when dealing with the system or simply did not know there were security policies. They also found that

these were not isolated incidences, security measure were either completely lacking or lacking one way or another.

4.2 Code Injection

Attacking a target system through code injection is one of the most common and potentially most dangerous types of attack on modern computer systems. Such attacks work by inserting and executing arbitrary code on the victim machines. Generally the objective is to get control of the machine's program counter (PC). This enables the attacker to change the flow of instruction execution and cause the PC to execute malicious code inserted by the attacker himself. This injected code could be source code for an interpreted scripting-language, intermediate byte-code, or natively-executable machine code. Such code may be designed for instant attack, where it executes instantly (e.g. Stealing a user's current session information or executing a modified SQL query), or it may execute at a later time.

Another common way by which attackers target systems, is by exploiting the code that accepts input from a source. The input could be a simple string of characters. But if not checked for issues such as stack and heap overflows, it can cause extensive damage. There have been instances where buffer overflows was caused by maliciously prepared embedded images or audio files that the operating system failed to validate and that enabled an attacker to execute arbitrary code on the system.

4.3 Inefficiency of encryption

There are many methods to encrypt data passing through a network, whether within a company or over the internet. One way to protect data going through an unsecure channel is using keys of public and private combination. The Public key encryption method uses two mathematically related keys. The data is encrypted with one key and decrypted with the other. Thus it is an asymmetric process. If a person sends out a message that has been encrypted using the public key, then only the person with the private can decrypt and read the valid message. On the other hand, Private Key encryption uses a single key. This key is known only to the two parties involved in the transaction. This process is computationally less intensive but it requires that the private key should remain private.

These methods may be suggested as a technique to solve our problems of susceptibility to an external attack. But encryption is not fool proof in itself. It is possible for an attacker to intercept an encrypted packet and alter its contents before passing it on. This results in scrambled data. There are other techniques (or combinations of them) such as hashing that is used to secure the data transmission. But these elaborate methods are taxing for most embedded systems. The main reason is the high computational capability required for encryption. The fast processor and high memory resources required for such auxiliary activities are prohibitive for many embedded systems which have very limited processor and memory. Another reason why encryption cannot be used is illustrated by the following example. Consider a gaming console that has access to the internet. If the system is to allow access to third part games, then we could not possibly use private key encryption. Public key encryption, on the other hand leaves the system open to attack from anybody pretending to be a game developer.[6]

4.4 Exploiting Shared Memory

The early RTOSs lacked a memory management unit. They used a flat shared memory model for inter-process communication. This method can be explained by assuming that the shared memory is like a white board in a room full of people. The people use the board to pass messages to each other. Person A comes in and writes a message for a person B on the white board. However another person can come and modify this message before person B reads it. If this were to happen in a computer system, it would not be difficult to image how the whole system could crash. This method does work but it

requires many system calls and careful management. But it is clear how a malicious process can corrupt the contents of the memory easily.

Since there is no restriction on the reading of memory, a malicious program may read sensitive information and compromise privacy. For instance, it can look for ASCII-data patterns in memory that could correspond to passwords or personal information. The program could then use the network link it came in to send that sensitive information to a third party.[6]

4.5 Priority Inversion

RTOS today need to process multitudes of jobs and each of them might share resources like memory or I/O. Limited resources like these cannot be share the same way that memory is shared. Since some processes are inherently more important than others, they are put into a priority, where higher priority processes could force a context switch with a lower priority process. Another problem with shared memory is that a process needs to be able to completely write a block of memory and cannot be interrupted while doing so; this means that other processes that want to use the memory cannot do so else it would cause memory inconsistency. The concept is called mutual exclusion, where only one process can use a resource at a time.

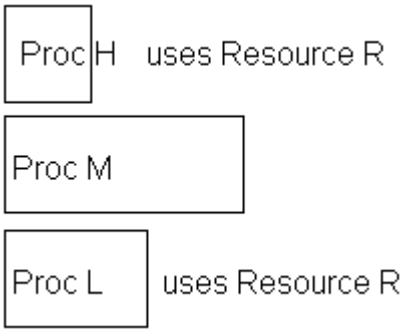

Figure 2. Proc size corresponds with job length.

Mutual exclusion and process priority tries to solve the problem of data sharing and processor sharing, respectively. The combination of both creates another problem called priority inversion. This happens in a system of three or more priority levels.

For example, figure 2 shows three processes; process *H* has the highest priority, then process *M*, then process *L*. process *H* and process *L* share the same resource *R* so one has to wait for the other to finish. In a normal situation, process *H* would finish with resource *R*, then let go of resource *R* and processor control, process *L* would take over with resource *R*, finish, and process *H* would take over again. An unwanted situation occurs when process *M* forces context switch because process *L* is of lower priority.

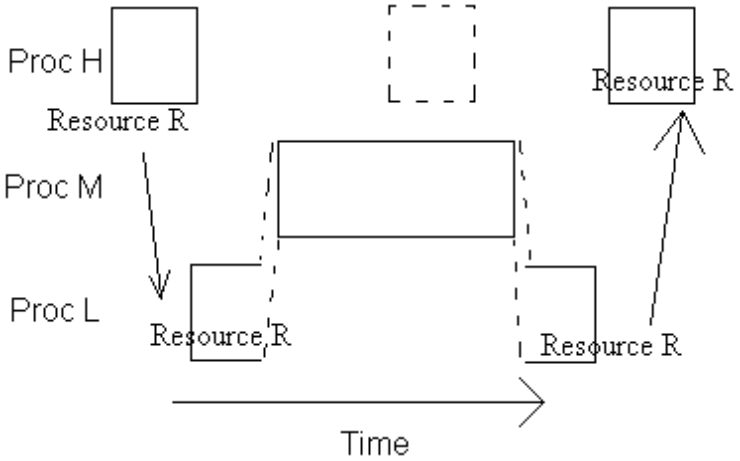

Figure 3. Proc H cannot force context switch on Proc M because it needs Resource R that Proc L is holding.

Shown in figure 3, while process *L* is in execution, process *M* forces context switch, when process *H* tries to take control and use *R* it cannot because process *L* has resource *R*, so process *H* is forced to wait until process *M* is done execution and also process *L*'s execution before it can use resource *R* again. This effectively 'inverts' the priority of processes because process *H* is the last to execute.

4.6 Denial of Service Attacks

There are several ways in which a malicious piece of code can take down the system. Suppose a system consists of two application programs each running one task (with the same priority, using Round Robin technique). Under ideal conditions, the two processes share the CPU's time, each taking 50%. But due to bad design or malicious intent, if one of the applications spawns 100 additional tasks, there are 101 tasks sharing time with the good application. Thus the good application is

starved of CPU access. It gets less than 1% of CPU access instead of 50%.

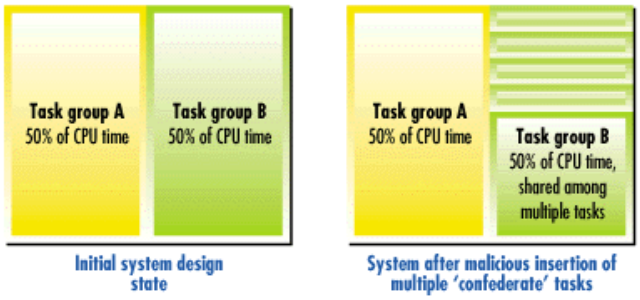

Figure 4. Malicious code causes CPU starvation (Source: Green Hills)

Denial of Service occurs when one part of the system "hogs" system resources such that other parts of the system cannot operate normally. For example, if one program hangs in an endless loop, other programs are starved of CPU time. Or perhaps if one program allocates too much memory, it starves other programs' memory requirement. Denial of service can be caused by bad programming or by a virus or hacker accessing the system externally.

4.7 Attacking Inter-Process Communication

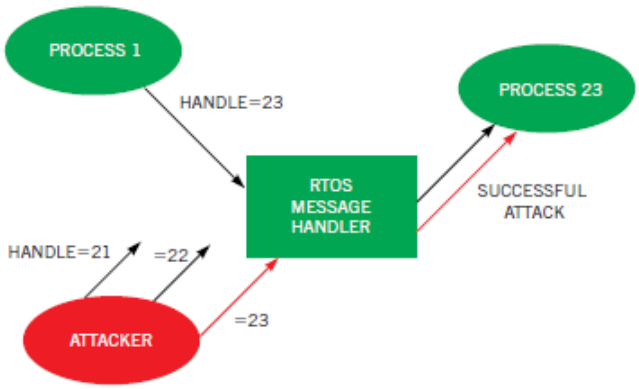

Figure 5. Malicious process disrupting the behavior of embedded system (Ref: www.opengroup.org)

To transfer information from one process to another, most RTOSs use messaging queues. The sending process has to make a system call to the RTOS. A handle is an identifying number to indicate which process is supposed to receive that message. The system call allocates a pointer to the memory location where the information is present and a handle for the destination. Most RTOSs have no way to ensure that an incoming message is from a legitimate source. As long as the handle is a valid value, the request will be completed. This allows a simple program that sends random messages to all possible destination handles to effectively scramble a system's inter-process communications. Figure 5 gives an illustration of this problem.

5.0. Common Approaches and Solutions

Since embedded systems resemble more and more like computer systems, some solutions are the similar, but not all. There are problems unique to embedded systems and SCADA because of their greater requirements like uptime, critical functions, deadline, and power consumptions.

5.1 Enforcement of Security Policies

Many SCADA systems could simply be made more secure if their companies have a competent IT staff that did regular checks on their SCADA network. There are tools online to check for vulnerabilities and there is no excuse not to do so. Management of these companies should obviously step up and require their employees to follow protocol when dealing with their SCADA system. Since SCADA systems now often use common computer networking, there should be no problem in making SCADA as secure as computer networks. Venders could help by requiring setup procedures to include mandatory password change and reset when customers buy their product. Companies could forbid employees from plugging in USB drives from one computer to another, to stop virus spread. SCADA system should also keep track of all employees' accounts and all employees should have their own account. Private networks should obviously be private and a computer should never have access to multiple networks in a company. Finally, former

employees' accounts should be deleted from the system immediately thus preventing retaliation.

5.2 Improving Existing SCADA

Old SCADA should not get a pass at being insecure simply because they are old. SCADA could be partially upgraded to have security software between employee monitors and machinery. Readouts from many machineries could go directly into an Embedded controller with security software that sends secure information to employee monitors. The physical setup should then be locked and made hard to intrude physically.

5.3 Dynamic Security Policies through Heuristics

To prevent security breaches even in newer secured systems. A SCADA network can be custom tuned by understanding the context of its environment. If the SCADA system were deployed in a natural gas production environment, a computer could keep track of all the sensors and disallow any commands that would jeopardize the stability of the system. For example, a computer monitoring various valves and tanks will see the correlation between valves and tank pressure. If an employee enters any command that could initiate a failure, the computer directly controlling the values should then reject the command. The invalid command should then be recorded in a central database.

The centralized database should have multi-level user policies that tiers employees into different access levels. Low level access would either be read-only or minor changes to non-critical systems. More access is granted to higher and higher levels. The specifics of access would depend on the environment it is deployed. The database would also keep track of employee commands and generate a history of it to be viewed by managers. The database itself should also allow or deny changes to the system depending on past behaviors of the employee. If an employee is denied a valid change to the system, a manager could have it approved with higher access. This would also be recorded and put against the employee's command trend. If this type of change to the system happens often enough, then the security policy would change to allow that employee to have permanent access that system change, without manager's explicit approval each time.

This system will prevent intruders from sending in fatal commands because even if an intruder gains access to the computer. The computer itself will reject the command. If the intruder was able to fake an employee's authorization, the database heuristic would pick up on the peculiar command and reject it.

5.4 Use of Privilege Levels

The use of privilege levels gives the RTOS a tool for preventing malicious programs from seriously affecting system operation. Legitimate programs usually do not need to alter any system-level operation, such as disabling interrupts. The operating system reserves this sort of system level access for only certain programs. Usually, these are the programs that are present during the boot sequence .Other programs (such as application programs downloaded from the network) are not given enough privileges to modify functions that can potentially disable the system.

Such a mechanism can be implemented by the use of "rings". These rings represent the various privilege levels in a system, and decide what a program running at a particular privilege level can or cannot do. Typically a system has a minimum of two rings. The inner ring is the kernel mode. It has least protection and can access most resources. The OS runs in this mode during start up. The outer ring corresponds to the User Space and is used mostly for applications. It has the maximum level of protection and minimum access to resources.

The basic idea of memory protection to solve the problem of unwarranted memory access is to isolate the process from each other and from the OS. The memory management unit forms

an important part of this solution. The MMU associates certain pages of the physical memory with a virtual address space for a particular program. A task can only access the memory that has been mapped to its virtual addresses. Thus the system can prevent one program from writing into the memory of another program. This is achieved through a hardware mechanism that can establish multiple address spaces, and also detect if a program tries to read or write outside of its assigned address space. Full virtual memory functionality can be used to protect memory but can slow the operating system down depending on what speed requirements it needs.

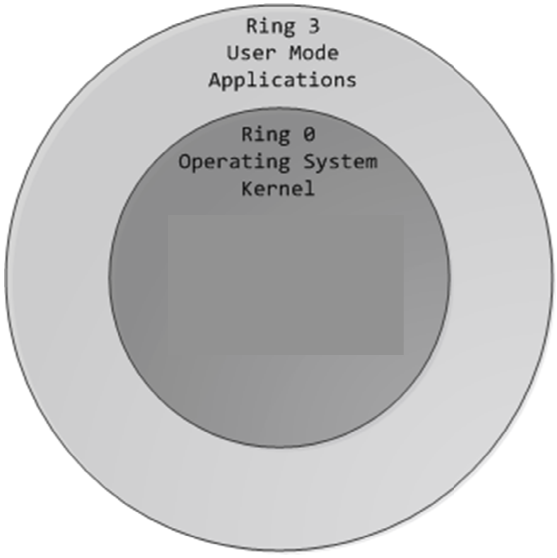

Figure 6. Operating System Protection Rings

5.6 Priority Inheritance Protocol

The way to prevent priority inversion is to not let it happen. Priority *L* has the lowest priority, then process *M*, then Process *H* with the highest priority. When process *L* is running with resource *R*, it inherits the priority, if higher, of Process *H*. This means that Process *L* will have high priority so process *M* cannot force a context switch on process *L*. Process *L* will finish with resource *R* and revert to the lowest priority state. Process *H* can then come in and take resource *R* or force context switch on process *M* if it gets processor time first.

5.7 MILS (Multiple Independent Levels of Security)

It is an architecture that defines a layered approach to security. Unlike monolithic kernels that perform all trusted functions for a secure operating system, MILS introduces the concept of a separation kernel to implement a set of functional security policies that regulate the information flow.

The MILS architecture provides a reusable formal framework for high assurance system security in embedded systems. The separation kernel is the base layer of the system and creates separate MILS process spaces (partitions), enforcing data separation and information flow control within a single microprocessor. The Middleware layer provides services such as resource allocation, object-oriented inter-partition or real-time data distribution services. Middleware services are concerned about end-to-end secure inter-object message flow. The Applications layer provides application specific security functions such as Firewalls, etc. The policy enforcements by the Micro kernel, Middleware and Application are non-bypassable, tamper-proof, evaluatable and are always invoked. The advantages of MILS separation kernel are that it is possible to know for each object, where the inputs came from and where the outputs are going. The Data for each object remains private. Additionally, each layer may be evaluated and enhanced separately without affecting other layers. Once the separation kernel is proven and secure, higher level secure software, such as a secure communications mechanism, web server or file system, can be layered on top.

A system built on such a concept requires certain hardware support. A few of the many factors that need attention before such a system can be considered are given below.

- Processing power: The real time system must have sufficient computational capability to meet the worst case timing requirement of the system.
- Privileged Mode: Some instructions are executed only by the Separation Kernel, so the system must provide for a privileged mode.

- Memory Management Unit: The MMU provides separation of address spaces between the partitions. Data isolation or damage limitation is not possible without hardware support for separation. The processor must have access to the required memory resources and must provide the Separation Kernel with the ability to restrict partition access to memory.
- Instruction Traps: The processor must have some mechanism to transfer control to the SK if a partition attempts to execute a privileged or invalid operation.

These are basic processor features and are generally available on many commercial microprocessors.

In this way, we see that through separation, we can develop a hierarchical model of security services. Each layer of this model utilizes the security services of a lower layer to provide a new security functionality that can be used by the higher layers. Each layer is responsible for only its own security. The separation kernel concept is powerful as it allows software with different security requirements to run on a single processor. For example, an application containing classified data and algorithms can safely occupy one partition while another partition is connected to the unclassified internet.

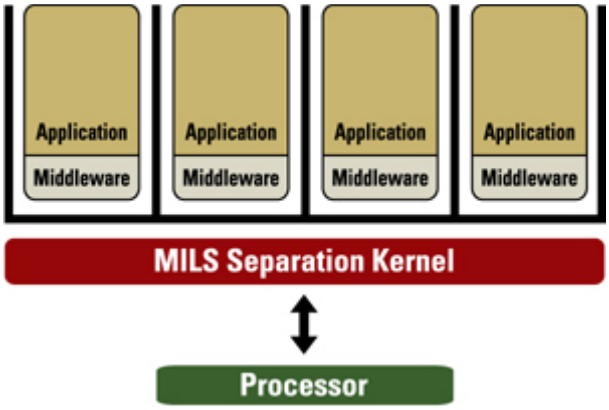

Figure 7. The use of Separation Kernel to improve security (Source: www.ois.com)

This can lead to enormous cost savings in product development as it can allow legacy operating systems (such as Windows and Linux) and non-critical applications to run in secure partitions on a processor. This is particularly important for large scale embedded systems with many people-years invested in existing code.[7]

6.0 Commercial System Design

Embedded systems are everywhere in the modern age, and many companies provide services for virtually any needs. The embedded system could be configured for anything from controlling lighting in your house to controlling the power grid.

6.1 Green Hills Software, Inc.

This company offers wide variety of RTOS solutions depending on your needs. Their RTOS range high power with security standard to micro kernels that can boot up in 1500 cycles. Their INTEGRITY RTOS is certified by NSA-managed NIAP labs to EAL6+ High Robustness. EAL evaluation is how secure a system is regarding the Common Criteria, an international standard. It also has hardware memory to protect itself from incorrect operation or tampering. Memory is divided into separate secure partitions and none can access each other, the only way for them to talk to each other is through the kernel by message passing. They also offer cheap alternative to their INTEGRITY line with velOSity RTOS. It features very short interrupt response time that is statically known.[3]

Some of their better known clients are Toyota, NASA, Lockheed Martin, Ford, and Northrop Grumman. Toyota Prius uses their RTOS for power train, drive train, power steering, anti-lock brakes, air bags, body control, and electric motor control system. NASA used their INTEGRITY RTOS for the Orion crew exploration vehicle; this included the spacecraft's avionics systems, flight control module, communications adapter, and backup emergency controller. Lockheed Martin's Joint Strike Fighter uses INTEGRITY to develop its safety-critical and security-critical software. Ford uses Green Hill's compiler and

instruction set simulator to develop control software for Lincoln Aviator. Northrop Grumman uses INTEGRITY for Airbus A380's inertial navigation system.[3]

6.2 LynuxWorks, Inc.

This company offers its own RTOS called LynxOS. They offer several variants of it and some conform to the POSIX standard and the DO-178B standard. The LynxOS Embedded RTOS is a hard RTOS that is used when a highly reliable system is needed. It includes support for networking. The LynxOS-SE RTOS features Time and Space Partition for Fault Containment. It makes it impossible for one partition to interfere with events on another. Each partition runs as a virtual machine with its own resources. Partitioning each process allows them to have fixed time-slices with the processor so the system is predictable at all times. LynxSecure is a hypervisor and separation kernel that lets multiple operating systems run on top to allow guest OS to run their applications. This lets clients run wide variety of applications on a single system. LynxSecure isolates each virtual instance by giving them a partition of memory, CPU time, and I/O peripherals. BlueCat Linux is a embedded Linux operating system based on Linux 2.6. It has preemption points in the kernel so user can suspend processes and start a higher priority process. It also has a custom scheduler that speed up selected task for execution. Applications written for BlueCat can be moved to LynxOS if user finds out they need a hard RTOS.[4]

Some of their better known clients are Boeing, U.S. Navy, airports, Airbus, John Deere, military, and U.S. Army. Boeing uses LynxOS for their cabin services system on Boeing 777. U.S. Navy war ship DDG-1000 uses LynxOS in multiple areas like their interfaces of the ship, the missile launching equipment, the ship's propulsion equipment, and communications equipment. U.S. Navy also uses LynxSecure for onboard computer systems where there is a high security requirement for all military equipment. Airports around the world uses Common Automated Radar Terminal Systems (ARTS) based on LynxOS to direct traffic. Airbus used LynxOS in A380 superjumbo jet for its Ethernet onboard that connects 25 PowerPCs together working in parallel. John Deere used iRbot's technology to create R-Gator, an unmanned vehicle that is used in dangerous military missions acting as unmanned scout, perimeter guard, supply carrier and more. iRobot used BlueCat Embedded Linux to power the robotics control, navigation, and object avoidance system. The Medium Extended Air Defense System (MEADS), used in United States and Germany, uses Lynx OS-SE for networking and network security. It uses LynxOS for its radar system. U.S. Army used LynxOS for onboard processing for the Crusader self-propelled howitzer. It also used LynxOS for shell ballistic calculation and graphical user interface and real-time equipment control systems.[4]

6.3 Wind River, Inc.

This company main RTOS product is called VxWorks. VxWorks is supported by a number of processors including the x86 family, MIPS, PowerPC, FreeScale, ColdFire, Intel i960, SH-4, etc. The VxWorks RTOS offers enhanced security features through the use of a Memory Management Unit and good partitioning schemes. The OS works on a concept of protection domains. It provides hardware enforced memory protection. The single flat memory physical address space is extended to multiple virtual address partitions for running different applications. The kernel has its own domain. A developer loads an application with resources such as memory, tasks, queues, and semaphores into a protection domain, thereby isolating and protecting the application from applications in other protection domains. The protection domain also defines the basis for automated resource reclamation. The protection domains can be created either at system startup, or dynamically at run time, to encapsulate resources within a system. Some new versions of the VxWorks AE provide for

temporal partitioning, which allows users to control the amount of processor time the OS allocates for each application it runs.

VxWorks has been widely used in variety of devices from aerospace and defense applications to networking and consumer electronics gear, robotics and industrial applications, precision medical instruments, and navigation and telematics systems in automobiles. The systems have been successfully deployed in millions of devices by leading companies worldwide. Smiths Aerospace relied on Wind River Platform for Safety Critical ARINC 653 to help build out its Boeing 787 Dreamliner common core system. NASA used Wind River VxWorks real-time operating system to develop software packages for the Mars Exploration Rovers. Motorola uses it for enhanced, interactive set-top box design. Companies such as ABB and Honda have used VxWorks in their industrial robots.VxWorks RTOS have been used in many more applications involving industrial automation, building automation, medical, transportation, automotive telematics, and small-footprint consumer devices

6.4 Microsoft, Inc.

Microsoft offers a family of operating systems under the name Windows Embedded. The Windows Embedded CE (also known as Windows Embedded Compact) is a componentized real time operating system. The latest version Embedded CE 6.0 offers an open scalable, 32 bit operating system that integrates reliable, real-time capabilities with advanced Windows technologies. It provides a hard real-time, small-footprint OS with a redesigned kernel and embedded-specific development tools. It has a modular design with a specialized kernel that can run in under 1 MB of memory. It is available for ARM, MIPS, SuperH and x86 architectures. Microsoft also makes available a specialized version of Windows Embedded Compact, known as Windows Mobile, for use in mobile phones. It is a customized image of Windows Embedded Compact along with specialized modules for use in Mobile phones. Windows Embedded is used in a variety of devices ranging from small footprint, real-time devices to Point of Sale devices like kiosks.

Windows Embedded is widely used in devices ranging from portable digital picture frames, GPS devices, Portable Media players, Set Top Boxes, Remote Metering, VOIP phones, ATMs, Industrial Controls, etc. The advantages of using Windows Embedded CE include the ability to use familiar tools like Visual Studio and also extend your embedded device by connecting to Windows PCs, servers and online services.

7.0 Some Known Cases of Security Lapse

The pervasion of embedded systems in so many aspects of our day to day life has also brought to light various instances, where the security breach in RTOS can have cause great impact on our lives. Security lapse in any of these systems that we depend on so implicitly can potentially endanger our privacy, safety and well being.

7.1 Security in Vehicle Networks

A security analysis of a car done by University of Washington and University of California San Diego found that modern cars have many security faults within the car's network. Incredibly inept, the car's network has many faults; even where there was security, it was not enforced. The car uses a control area network, where each control unit talks to each other by broadcasting to all other control units. The control units themselves decide if they should react to the message packets. This makes it very easy for intruders to listen in on the messages. Another fault is that the packets sent has no identification, each control unit will accept messages without knowing if the message generated is from a valid unit in the car. The firmware of the control units at many times demonstrated it could be reflashed with invalid or edited firmware. The standards that it put in place to mitigate system failure from

intrusion was not enforced. For example, the researchers were able to reflash the main control unit while the car was in motion. Control unit was stated to reject reflashing attempts while vehicle is in motion. Packet protection was so bad that an intruder could enter random to semi-random packets and it will fault the vehicle. This was done to reverse-engineer packets to discover which packets performed which task. The high speed and low speed networks comes together and can attack each other if a control unit connected to both is compromised. The instrument panel could all be intruded to give wrong speed reading, full level, and random messages. The radio controls on the center console could be overwritten. An intruder could keep the radio volume at maximum no matter what the driver tries to do. Even worse is that the researchers could adjust all of the engine functions, such as resetting the engine crank shaft sensor. Disable the engine so it knocks excessively when restarted. Fake an airbag deploy signal so the engine won't start. The brakes could be individually turned on and stop the car without driver intervention. A denial of service attack could also be performed on the network to disrupt communication between each control unit. The control units could also be programmed to erase itself to erase the evidence of being tampered. The conclusion drawn by the researcher was that the attacks were easy, so much damage could be done after getting into the car network, and how the control access was not enforce evenly throughout the system.[2]

7.2 Risks in SCADA systems

The risks that SCADA systems face can be due to an intentional attack to gain unauthorized access to the control system or it could be unintentional attack caused due to inadequate testing, system failure, etc.

There are multiple examples of both types of attacks on SCADA systems in the recent years. In March 2008, the Hatch Nuclear Power Plant in Georgia was forced to shut down after a software update to the plant's business network. The business network was in two-way communication with the plant's SCADA network. The software update synchronized information on both systems. After a reboot, the SCADA safety systems sensors detected that the water level in the cooling systems was dangerously low and this triggered an automatic shutdown. Although this event did not cause any public harm, but the company lost a great deal of money in revenue and had to spend a lot of money to get the plant back in running position. This incident displays how insufficient testing of the systems can cause havoc. The engineers in this case did not know that the software update to the business network would synchronize with the SCADA network and thus cause such consequences.

There was another instance in January 2003, where the Sobig computer virus infected the computer responsible for controlling the signaling system for trains on the east coast. The Sobig virus opens a back door that lets a hacker gain access without being detected. The virus infected the systems at CSX Corp.'s headquarters, causing a temporary shutdown of signaling and dispatching systems. Various trains including long distance were delayed for two to four hours.[10]

7.3 Attacks on Mobile phones

The proliferation of unlimited data plans, open networks and readily downloadable applications make the mobile phone segment a big profit potential for hackers and spammers. The capability of mobile devices is significantly advanced than those in the past. This advancement in technology is very beneficial but it also brings its own share of security threats. There have been increasing numbers of instances where attacks have caused the device to collapse because the operating system was compromised. Smart phones are particularly susceptible because since they are internet end points, they can be compromised the same way as PCs by worms, viruses or Trojans. Also, mobile users tend to be less hesitant than computer users about clicking on links, enabling SMS phishers to

gain information or send malware via a link in a legitimate-looking text message.
Evidently, the rush to get new software into the market place results is deficient security testing and sub standard programs are marked off as acceptable to be deployed.

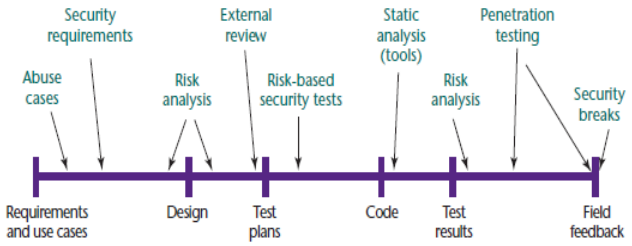

Figure 8. Software Security best practices applied to Waterfall model (Ref: www.cigital.com)

8.0 LESSONS LEARNED

There are many lessons that could be learned over the years. The security of a system starts at its development phase. It is not something that can be introduced to a system arbitrarily. Systems not built with security as a primary consideration, are more susceptible to attack and failure than those that had security as one of the design requirements. The security of an RTOS starts at its design and development phase. In this way, software professionals attempt to build software that can withstand attack proactively. Good programming techniques are inarguably important in bolstering an application against attacks. The programmers should take care to avoid common problems such as buffer overflows and race conditions. The program should be tested vigorously from security perspective for different possible attack methods before being deployed for public use. The critical programs should be debugged to maximize their security even in case of attack.

To protect the RTOS from network related attacks, the common methods of firewalls and intrusion detection systems should be deployed. The cryptographic techniques developed for Wireless Sensor Networks which suffer from similar constraints of limited computational capability and low transmission rate of data, can be applied to Embedded Systems. The systems should be monitored for security breaks and the knowledge gained from this should be cycled back to incorporate additional features and measure to counter similar attacks in the future. The networks can also employ the use of monitoring devices that regulates the use of resources by the different processes. It acts like a centralized system that is invoked by all processes when they want to access a resource. The process should have sufficient privileges for the resource. Such a system prevents malicious programs from exhausting the system's resources.

Effective security requires the use of both proper technology as well as competent management. Security cannot be achieved by one time implementation. Even the most advanced and proven technologies require constant monitoring. Like all computer systems, the hardware and software should be periodically updated to the latest and most stable definitions. Also, it has been observed that regular audits by third parties have exposed many bad practices.

9.0 CONCLUSION

RTOS's very specific time constraint made it a primary choice when upgrading existing or new SCADA system. These RTOS's inside embedded systems are being adopted at an amazing pace. The rise of networking and the similarity between modern embedded systems and computers made them very easy to be connected through existing computer networks. This enhances the systems' versatility and production value. Embedded systems' inherent reliability and increasing capability garners its widespread use, but this comes at a price. Security is lacking in many SCADA in use. Its lack of security is derive from the fact that SCADA are omnipresent and obscure. This was use as a defense against intrusion but it is not enough. SCADA's increasing similarity with computer network makes them vulnerable to attacks common in computer networks. This combines with their

importance in society highlights the critical need to implement security measures to protect SCADA and the embedded systems that forms them. Their use only gets more widespread as time progresses. It is very possible to protect these systems with modern technologies and solutions found in computer networks. Coupled with users following security protocols, SCADA can be made just as secure as modern computer networks.

10.0 REFERENCES

[1] Goering, Richard. "Getting a Lock on RTOS Security." *EETimes.com*. EE Times, 9 Apr. 2007. Web. 16 May 2010. <http://www.eetimes.com/news/design/showArticle.jhtml?articleID=198800770>.

[2] University of Washington and University of California San Diego. *Experimental Security Analysis of a Modern Automobile*. 16 May 2010. Web. 18 May 2010. <http://www.autosec.org/pubs/cars-oakland2010.pdf>.

[3] *Real-Time Operating Systems (RTOS), Embedded Development Tools, Optimizing Compilers, IDE Tools, Debuggers - Green Hills Software*. Web. 19 May 2010. <http://www.ghs.com/>.

[4] *RTOS for embedded real-time systems from LynuxWorks – LynuxWorks inc*. Web. 19 May 2010. <http://www.lynuxworks.com/>.

[5] Maynor, David, and Robert Graham. "SCADA Security and Terrorism: We're Not Crying Wolf!" Lecture. Black Hat Federal 2006. 1800 Jefferson Davis Highway, Arlington, Virginia. *Black Hat Archives*. Web. 18 May 2010. < http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Maynor-Graham-up.pdf>.

[6] Monkman, Robert. "Enhancing Embedded Security." *EDN.com* 17 Oct. 2002: 61-66. <http://www.opengroup.org/press/articles/17oct2002-Enhancing%20Embedded%20Security.pdf>

[7] Alves-Foss, Jim, Scott W. Harrison, Paul Oman, and Carol Taylor. *The MILS Architecture for High-Assurance Embedded Systems*. 2005. International Journal of Embedded Systems. <http://www.csds.uidaho.edu/papers/Alves-Foss06a.pdf>

[8] Igure, Vinay M., Sean A. Laughter, and Ronald D. Williams. "Security Issues in SCADA Networks."*Computers & Security* 25 (2006). < http://www1.elsevier.com/homepage/saf/infosecurity/research/1206_scada.pdf>

[9] Smith, Christopher. "RTOS - Securing the Future of Embedded Systems." *Mtemag.com* 01 Sept. 2009. Web. < http://www.mtemag.com/ArticleItem.aspx?Cont_Title=RTOS+-+Securing+the+future+of+embedded+systems>

[10] Tsang Rose. Cyberthreats, Vulnerabilities and Attacks on SCADA Networks. <http://gspp.berkeley.edu/iths/Tsang_SCADA%20Attacks.pdf>